Department of Computer Science
Series of Publications A
Report A-2016-6

Combinatorial Algorithms with Applications in Learning Graphical Models

Juho-Kustaa Kangas

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium CK112, Exactum, Gustaf Hällströmin katu 2b, on December 9th, 2016, at 2 p.m.

University of Helsinki
Finland

Supervisors
Mikko Koivisto, University of Helsinki, Finland
Matti Järvisalo, University of Helsinki, Finland

Pre-examiners
Seiya Imoto, University of Tokyo, Japan
Cassio de Campos, Queen's University Belfast, United Kingdom

Opponent
James Cussens, University of York, United Kingdom

Custos
Petri Myllymäki, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911, telefax: +358 9 876 4314

Copyright © 2016 Juho-Kustaa Kangas
ISSN 1238-8645
ISBN 978-951-51-2724-2 (paperback)
ISBN 978-951-51-2725-9 (PDF)
Computing Reviews (1998) Classification: F.2.1, F.2.2, G.2.1, G.3, I.2.6, I.2.8
Helsinki 2016
Unigrafia

Combinatorial Algorithms with Applications in Learning Graphical Models

Juho-Kustaa Kangas

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
juho-kustaa.kangas@helsinki.fi
http://www.cs.helsinki.fi/u/jwkangas/

PhD Thesis, Series of Publications A, Report A-2016-6
Helsinki, November 2016, 66+90 pages
ISSN 1238-8645
ISBN 978-951-51-2724-2 (paperback)
ISBN 978-951-51-2725-9 (PDF)

Abstract

Graphical models are a framework for representing joint distributions over random variables. By capturing the structure of conditional independencies between the variables, a graphical model can express the distribution in a concise factored form that is often efficient to store and reason about.

As constructing graphical models by hand is often infeasible, a lot of work has been devoted to learning them automatically from observational data. Of particular interest is the so-called structure learning problem, of finding a graph that encodes the structure of probabilistic dependencies. Once the learner has decided what constitutes a good fit to the data, the task of finding optimal structures typically involves solving an NP-hard problem of combinatorial optimization. While first algorithms for structure learning thus resorted to local search, there has been a growing interest in solving the problem to a global optimum. Indeed, during the past decade multiple exact algorithms have been proposed that are guaranteed to find optimal structures for the family of Bayesian networks, while first steps have been taken for the family of decomposable graphical models.

This thesis presents combinatorial algorithms and analytical results with applications in the structure learning problem. For decomposable models, we present exact algorithms for the so-called full Bayesian approach, which involves not only finding individual structures of good fit but also computing posterior expectations of graph features, either by exact computation or via Monte Carlo methods.

For Bayesian networks, we study the empirical hardness of the structure learning problem, with the aim of being able to predict the running time of various structure learning algorithms on a given problem instance. As a result, we obtain a hybrid algorithm that effectively combines the best-case performance of multiple existing techniques.

Lastly, we study two combinatorial problems of wider interest with relevance in structure learning. First, we present algorithms for counting linear extensions of partially ordered sets, which is required to correct bias in MCMC methods for sampling Bayesian network structures. Second, we give results in the extremal combinatorics of connected vertex sets, whose number bounds the running time of certain algorithms for structure learning and various other problems.

Computing Reviews (1998) Categories and Subject Descriptors:

F.2.1	[Analysis of Algorithms and Problem Complexity] Numerical Algorithms and Problems
F.2.2	[Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems
G.2.1	[Discrete Mathematics] Combinatorics – combinatorial algorithms, counting problems, recurrences and difference equations
G.3	[Probability and Statistics] multivariate statistics, probabilistic algorithms (including Monte Carlo)
I.2.6	[Learning] knowledge acquisition
I.2.8	[Problem Solving, Control Methods, and Search] dynamic programming, heuristic methods

General Terms:
algorithms, design, experimentation, theory

Additional Key Words and Phrases:
Bayesian networks, connected subgraphs, decomposable models, empirical hardness, exact algorithms, linear extensions, structure learning

Acknowledgements

First of all, I would like to express my deepest gratitude to my advisors Mikko Koivisto and Matti Järvisalo for their patient guidance throughout my PhD research as well as to Petri Myllymäki, who has played various roles in making this work possible. Many thanks go also to my other colleagues and co-authors Teemu Hankala, Petteri Kaski, Janne Korhonen, Brandon Malone, Teppo Niinimäki, and Pekka Parviainen. It has been an utmost pleasure to work with and learn from all these brilliant people.

I thank also my pre-examiners Seiya Imoto and Cassio de Campos as well as the anonymous reviewers of the papers that make up this thesis for their helpful comments and suggestions.

I am immensely grateful for the financial support and guidance that I have received from the doctoral programmes Hecse and DoCS throughout my studies. I thank the DoCS steering committee and Pirjo Moen in particular for her role as research coordinator and for her valuable advice in finalizing this thesis. I extend my thanks further to the Department of Computer Science and Helsinki Institute for Information Technology for providing an excellent environment for research. I thank all of the staff and the many friends and acquaintances I have met here.

I am grateful also for my brief research visit at TU Vienna. I thank Stefan Szeider and all members of the Algorithms and Complexity Group for hosting me. In particular, I thank Eduard Eiben, Robert Ganian, and Sebastian Ordyniak for our time and work together.

Finally, I thank my parents and siblings and all other people who have supported me in life during these years.

Helsinki, November 2016
Juho-Kustaa Kangas

Contents

Original publications

This thesis is based on the following original publications, referred to as Papers I–VI in the text. The papers are reprinted at the end of the thesis.

I. Kustaa Kangas, Mikko Koivisto, and Teppo Niinimäki. Learning chordal Markov networks by dynamic programming. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 2357–2365. Curran Associates, Inc., 2014.

II. Kustaa Kangas, Teppo Niinimäki, and Mikko Koivisto. Averaging of decomposable graphs by dynamic programming and sampling. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 415–424. AUAI Press, 2015.

III. Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Predicting the hardness of learning Bayesian networks. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2460–2466. AAAI Press, 2014.

IV. Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Empirical hardness of Bayesian network structure learning. Under revision.

V. Kustaa Kangas, Teemu Hankala, Teppo Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 603–609. IJCAI/AAAI Press, 2016.

VI. Kustaa Kangas, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. On the number of connected sets in bounded degree graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 8747 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2014.

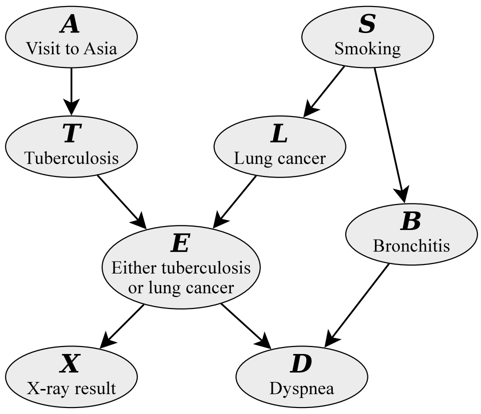
Chapter 1

Introduction

Reasoning in the presence of uncertainty is a challenging task that arises in various areas, requiring one to model interactions between attributes whose values are often unknown. A prime example is in medicine, where a doctor observes a set of symptoms in a patient and needs to determine the most likely diagnosis, often amongst many possible diseases. For measuring and reasoning about such uncertainty, by far the most common tool is the notion of *probability*. In this general framework the attributes of the domain, such as the presence of a particular symptom or disease, are expressed as *random variables* and the uncertainty about possible values is quantified as a *joint probability distribution* over the variables. This allows principled reasoning about the uncertainty, such as determining the probability that a patient with certain symptoms has a particular disease.

In order to reason about a distribution one first needs to represent it in some convenient form. A naive way would be to specify explicitly the probability of each possible outcome of the variables. An obvious drawback with this representation is that the number of possible outcomes scales exponentially in the number of variables, even in the simplest case where each variable may only take on two possible values. This quickly becomes infeasible in many real-life domains, which may contain hundreds or thousands of variables. On the other hand, such domains typically also contain many *independencies* between variables, which often allows one to represent the distribution in a more concise, factored form. The focus of this thesis is on *probabilistic graphical models* [60, 68, 90], which are a popular and flexible framework for representing such distributions efficiently.

A graphical model expresses the independencies of a distribution with a *graph* consisting of a set of *vertices* and a set of *edges* between pairs of vertices. Each vertex corresponds to a single random variable, and an edge between two variables represents a direct probabilistic dependency between



$$\begin{array}{ccc|c}
A & \perp & S, L, B & \\
S & \perp & T & \\
T & \perp & S, L, B & A \\
L & \perp & A, T, B & S \\
B & \perp & A, T, E, X & S \\
E & \perp & A, S, B & T, L \\
X & \perp & A, S, T, L, B, D & E \\
D & \perp & A, S, T, L, X & E, B
\end{array}$$

Figure 1.1: An artificial example [69] of a Bayesian network on eight binary-valued variables. Each variable indicates either the presence or absence of a disease (Tuberculosis, Lung cancer, Bronchitis), a possible risk factor (Visit to Asia, Smoking) or a symptom (X-ray result, Dyspnea). The edges are directed by assumed causality, with risk factors causing their respective diseases, and the diseases causing their respective symptoms. The network asserts that each variable is independent of its non-descendants given its parents. These independencies are listed next to the network.

them. The edges may be either directed or undirected, which gives rise to two commonly used families of graphical models, the directed models, more commonly known as Bayesian networks [90], and the undirected models, also known as Markov networks or Markov random fields [60, Ch. 4].

The graph is known as the *structure* of the graphical model, and it encodes a set of conditional independencies between the random variables. Figure 1.1 shows an example of a medical diagnostics network, which depicts the dependencies between diseases and their possible symptoms and risk factors. We can see, for instance, that whether a person has visited Asia or has bronchitis are implied to be (marginally) independent of each other, while the edge between B and D indicates a possible relation between having bronchitis and having dyspnea (shortness of breath). This aside, the edge reveals no further qualitative or quantitative information on the dependency. From the structure alone we cannot tell, say, whether having bronchitis makes one more or less likely to suffer from dyspnea.

In order to fully quantify all dependencies, we make use of the fact that the independencies guaranteed by the structure are equivalent to a certain factorization of the joint distribution. In a Bayesian network, for instance,

$\Pr(A=1)$
0.01

$\Pr(S=1)$
0.5

T	$\Pr(T=1 \mid A)$
1	0.05
0	0.01

T	L	$\Pr(E=1 \mid T, L)$
1	1	1
1	0	1
0	1	1
0	0	0

L	$\Pr(L=1 \mid S)$
1	0.1
0	0.01

B	$\Pr(B=1 \mid S)$
1	0.6
0	0.3

X	$\Pr(X=1 \mid E)$
1	0.98
0	0.05

E	B	$\Pr(D=1 \mid E, B)$
1	1	0.9
1	0	0.7
0	1	0.8
0	0	0.1

$$\begin{aligned}\Pr(A,S,T,L,B,E,X,D) &= \Pr(A)\ \Pr(S)\ \Pr(T \mid A)\ \Pr(L \mid S)\ \Pr(B \mid S) \\ &\quad \times \Pr(E \mid T,L)\ \Pr(X \mid E)\ \Pr(D \mid E,B)\end{aligned}$$

Figure 1.2: The factorization of the joint distribution over the Bayesian network structure of Figure 1.1 and the resulting local conditional probability distributions for each random variable.

the joint probability over all variables decomposes into a product of local distributions that specify the probability of every variable conditioned on the values of its parents. The numbers that specify these factors are called the *parameters* of the graphical model. Figure 1.2 shows the factorization for the example network as well as possible values of its parameters.

The graphical representation of the joint distribution has numerous advantages compared to an explicit description. First, it is typically much more concise and thus more feasible to store and manipulate. For instance, in the medical diagnostics example there are $2^8 = 256$ possible joint assignments of the variables, while specifying the local conditional probabilities only requires 18 parameters. Second, a graphical model is interpretable, as it allows one to directly observe dependencies between variables. Finally, suitably sparse graph structures admit efficient algorithms for answering probabilistic inference queries, such as: given that a patient has a positive X-ray result but no dyspnea, what is the probability that the patient has lung cancer?

The emergence of graphical models can be dated roughly back to the beginning of the 1900s, when graphical representations were first used in physics to model interactions between elementary particles. Throughout the century similar ideas arose independently in fields such as genetics, economics, statistics, and artificial intelligence. Following the influential theoretical work [90, 69] on graphical models and their use for probabilistic inference in the 1980s, they have since spread to numerous other areas.

Especially in early stages of their adoption, graphical models were often constructed by a domain expert with an understanding of how the variables are related. While construction by hand may be doable in simple domains, it is time consuming and becomes infeasible if the number of variables is very high, or if the domain is poorly understood or changes over time. Indeed, the lack of understanding of a phenomenon is often the main reason why one would want to describe it with a graphical model in the first place.

Even if a domain is not well understood, one typically has access to some amount of *data* consisting of multiple joint observations of the variables. For instance, in the medical diagnostics example an extract from a patient record might contain one data point or *sample* for each patient, detailing the status of all diseases, causes, and symptoms. Through statistical methods, it is possible to infer or *learn* the relationships between variables from the data and model them as a joint distribution on the variables. By treating a family of graphical models as a hypothesis space for the distribution, these methods allow one to automatically learn a graphical model that best explains or *fits* the data. The learning task is typically split into two separate phases, learning the structure and learning the parameters. If the structure has already been learned or is known beforehand, fitting its parameters to the data is considered a relatively easy problem, usually solved by maximum likelihood estimation or Bayesian methods (see e.g. [45, 60]). A significantly harder problem is the task of inferring the structure of conditional independencies, called *structure learning*. In this thesis we study the structure learning problem and its computational aspects for Bayesian networks and so-called decomposable models, which are the intersection of Bayesian networks and Markov networks.

There are two major and highly distinct approaches to learning the structure: the *constraint-based approach* and the *score-based approach*. In constraint-based learning [91, 113, 80, 102, 18] we aim to first infer the set of conditional independencies between the variables by carrying out a series of statistical independence tests. Once the independencies have been learned, a graph structure that corresponds to those independencies is constructed. This approach has the advantage of allowing principled treatment of unobserved variables but provides no means to assess the uncertainty about the learned structure.

This thesis focuses on score-based learning [17, 23, 46], where the learner assigns to each potential structure a real-valued *score* that measures how well the structure fits the data. The score can be defined in multiple ways but is typically based on the likelihood of the model combined with either a Bayesian prior belief or an information-theoretic penalization term. For

Bayesian networks such scores can usually be evaluated exactly, while for Markov networks this task appears computationally infeasible (see e.g. [60, Ch. 20]). Hence, the scope is usually restricted to learning *chordal* Markov networks, which are equivalent to the class of decomposable models.

Once the scoring criterion has been decided, the computational task of finding a structure with the highest score becomes a discrete search problem in the space of possible structures. Since the number of structures grows super-exponentially in the number of variables, exhaustive search in this space is infeasible outside of very small domains. Fortunately, commonly used scoring criteria admit a natural factorization of the score, which renders the search task a combinatorial optimization problem. Although the factorization avoids the enumeration of all structures, the search problem remains NP-hard for both general Bayesian networks [19] and decomposable models [104]. As a consequence, most early work on structure learning focused on developing local search algorithms [103, 20, 46, 67] that might often perform well but fail to provide guarantees about the quality of their output. Since last decade, however, there are has been a growing interest in developing *exact* algorithms that are able to find provably optimal structures. For Bayesian networks the first global search algorithms of this kind adopted a dynamic programming approach [85, 58, 101, 99] that is guaranteed to solve the problem in exponential rather than super-exponential time. More recently, this approach has been restated as a shortest-path problem, which can often be solved faster by employing various best-first heuristics [120]. Other approaches have included a branch-and-bound search in the space of cyclic graphs [29] as well as translating the problem into integer linear programming [27] or constraint programming [112]. For decomposable models, the first exact algorithms [25, 105] are likewise based on constraint optimization.

Besides finding a single "best" structure, score-based learning also admits the so-called full Bayesian approach, where the scoring function specifies a full *posterior distribution* over all structures, thus taking into account the uncertainty about the structure. Since representing the posterior distribution explicitly is not feasible or particularly useful, it is typically summarized by computing some interesting statistic, such as the posterior probability of all edges. For certain graph features this computation can be carried out exactly, while others can be estimated by sampling structures from the posterior. In particular, a variety of sampling schemes based on Markov chain Monte Carlo (MCMC) methods [1] have been proposed for both Bayesian networks [76, 38, 42, 37, 84] and decomposable models [39, 108, 24, 41].

1.1 Research questions and organization

This thesis presents advances in the structure learning problem, mainly in form of combinatorial algorithms for structure learning, as well as methods for expediting and analyzing existing algorithms. The thesis is based on six papers, which study four distinct questions.

Question I concerns structure learning in decomposable models. The current exact algorithms for this problem leverage the machinery of constraint optimization by encoding the problem as an instance of, say, maximum satisfiability [25] or integer linear programming [105], and then employ state-of-the-art algorithms for solving the resulting constraint optimization problem. While these algorithms can sometimes solve even large instances fast, they have no reasonable worst-case guarantees and scale reliably only for very small instances (around 10 variables). Further, they are only able to provide one or more highest-scoring structures and are not suited for the full Bayesian approach. Paper I addresses the first issue by presenting a dynamic programming algorithm that is guaranteed to solve the problem within exponential time and scales up to around 20 variables. Paper II addresses the second issue by adapting the algorithm for summation over all structures, thus enabling the computation of posterior probabilities, either exactly or by unbiased estimation via sampling.

Question II concerns structure learning in Bayesian networks, for which a variety of exact algorithms have already been proposed. As many of these algorithms are based on adaptive search strategies, their running times depend heavily on intricate structural properties of the problem instances, rather than simple parameters such as the number of variables. While all of the algorithms are guaranteed to find an optimal solution eventually, it is *a priori* not clear which one of them will finish first and how long it will take to run. Papers III and IV tackle this question by using machine learning methods to predict the running times on a given problem instance, based on an empirical evaluation of the algorithms and a variety of instance features. As a result, a hybrid algorithm is obtained that combines the best-case performance of multiple algorithms by always running the algorithm that is predicted to be the fastest on a given instance.

Question III concerns the fundamental combinatorial problem of counting linear extensions of a partially ordered set. This problem arises in a variety of applications, including bias correction in certain MCMC schemes for sampling Bayesian network structures [37, 84]. Although the problem is #P-complete [15], it can be solved efficiently for partial orders with a suitable structure. In particular, known counting methods can be fast on orders with a dense graph representation. Paper V complements such methods by presenting algorithms that are fast when the graph representation is sparse and thus particularly suitable for orders derived from Bayesian networks.

Question IV concerns the extremal combinatorics [51] of connected vertex sets in undirected graphs. The number of such sets bounds the time complexity of various graph algorithms, including a particular approach to structure learning in Bayesian networks [93] as well as certain algorithms presented in Paper V. Paper VI presents analytic results on the number of connected sets in graphs of bounded vertex degree.

The remainder of this thesis is organized as follows. Chapter 2 presents concepts that will be required throughout the thesis, including formal definitions of graphical models and the computational problems that will be considered in later chapters. The contributions of the thesis are presented in Chapters 3–6, which address Questions I–IV, in respective order. We conclude the presentation with a summary and discussion in Chapter 7. The original publications that the chapters are based on are printed at the end of this thesis.

1.2 Author contributions

The original publications were jointly written by their respective authors. Other contributions by the present author are as follows:

Papers I & II: The present author produced a formal proof of the main recurrence, devised and proved correctness of the sampling schemes jointly with Teppo Niinimäki, implemented all algorithms, and carried out the experiments.

Papers III & IV: The present author carried out the running time experiments and participated in discovery of new features and analysis of the results.

Paper V: The present author developed the recursive algorithm jointly with Teemu Hankala, implemented both algorithms, and carried out the experiments.

Paper VI: The present author considered the projection theorem in extended neighborhoods (suggested by previous research) and proved the novel upper and lower bounds.

Chapter 2

Preliminaries

In this chapter we give formal definitions of graphical models and the structure learning problem, which will be required in the following chapters. We also review existing approaches to structure learning.

2.1 Graphical models

Our interest is in modelling a joint distribution over n random variables, $X_1, X_2, \ldots, X_n$. The variables may be either discrete or continuous, though we will mainly focus on discrete settings. We denote the index set of the variables by $V = \{1, \ldots, n\}$, often identifying a variable X_v simply with the respective index $v \in V$. For each subset $S \subseteq V$ of the indices we denote by X_S the joint variable $(X_{v_1}, X_{v_2}, \ldots, X_{v_s})$, where $v_1, v_2, \ldots, v_s$ are the elements of S in increasing order.

Our objective is to express the joint distribution $\Pr(X_1, \ldots, X_n)$ more succinctly by exploiting conditional independencies between the variables. Graphical models are a flexible framework for capturing such independencies. We represent a graphical model as a pair (G, θ), where G is called the *structure* and θ is a vector of *parameters*. The structure is either a directed or undirected graph that encodes the conditional independencies of the distribution. Each vertex in the graph represents a random variable, and an edge between two variables corresponds to a (direct) probabilistic dependency between the variables. The independencies asserted by the structure imply a certain factorization of the distribution, and the parameters θ specify the exact values taken by the factors.

We proceed to give formal definitions of three widely used families of graphical models, the Bayesian networks and the Markov networks, based on directed and undirected graphs, respectively, as well as the so-called decomposable models, which are exactly the intersection of the first two. In later chapters we will focus mainly on the Bayesian networks and decomposable models.

2.1.1 Bayesian networks

A Bayesian network [90, 60] is based on a directed acyclic graph (DAG): a directed graph that contains no directed cycles. Let G be a DAG having the index set V as its vertex set. For each vertex (variable) $v \in V$, we denote by G_v the set of parents of v in G. Since every DAG has a topological sort, we may assume without loss of generality that the variables are numbered so that $G_v \subseteq \{1, 2, \ldots, v-1\}$ for each $v \in V$. For two variables $u, v \in V$ we call v a *descendant* of u if there is a directed path from u to v.

We say that G has the *local Markov property* if for all distinct $u, v \in V$ such that u is not a descendant of v it holds that

$$X_v \perp X_u \mid X_{G_v} ,$$

that is, all variables are conditionally independent of their non-descendants given their parents. From these independencies we can derive a factorization of the distribution. By the chain rule of probability we have that

$$\Pr(X_1, \ldots, X_n) = \prod_{v \in V} \Pr(X_v \mid X_1, X_2, \ldots, X_{v-1}) .$$

The assumed variable numbering implies that for each $v \in V$ the variables $X_1, X_2, \ldots, X_{v-1}$ contain the parents of X_v and none of its descendants. Thus the conditional probability $\Pr(X_v \mid X_1, X_2, \ldots, X_{v-1})$ simplifies to $\Pr(X_v \mid X_{G_v})$, which we call the *local conditional distribution* (LCD) of v. We may now express the distribution as

$$\Pr(X_1, \ldots, X_n) = \prod_{v \in V} \theta_v(X_v; X_{G_v}) , \tag{2.1}$$

where each $\theta_v(X_v; X_{G_v})$ is a function encoding the LCD of v. We identify θ_v with the vector of numbers that specify the distribution, called the *parameters*. For discrete variables we typically specify the probability of each outcome explicitly, and thus have one parameter for each joint assignment of $(X_v;\, X_{G_v})$.

We now define a *Bayesian network* as a pair (G, θ), where G is a DAG structure satisfying the local Markov property and $\theta = (\theta_1, \ldots, \theta_n)$ are the parameters of the LCDs.

2.1.2 Markov networks

A Markov network [60, Ch. 4] is based on an undirected graph. Though we will not work with general Markov networks, a brief formal definition will be useful for defining the class of decomposable models. To that end, let G be an undirected graph on the vertex set V. For disjoint sets $A, B, S \subseteq V$ we say that S *separates* A and B if every path between A and B in G contains a vertex in S. We now say that G has the (global) Markov property if for all disjoint vertex sets A, B, S such that S separates A and B it holds that

$$X_A \perp X_B \mid X_S \, .$$

As in Bayesian networks, the independencies guaranteed by the graph are equivalent to a certain factorization. We say that a nonempty set of vertices $C \subseteq V$ is a *clique* if it induces a complete subgraph of G. A clique is *maximal* if it is not a proper subset of another clique. If the joint distribution is strictly positive, then G has the global Markov property if and only if the probability factorizes as

$$\Pr(X_1, \ldots, X_n) = \frac{1}{Z} \prod_C \theta_C(X_C) \, , \tag{2.2}$$

where Z is a constant, C runs over the maximal cliques $C_1, \ldots, C_m$ of G, and each θ_C is a positive function [60]. The functions θ_C are commonly called *factors* or *clique potentials*. Although they are analogous to LCDs in Bayesian networks, the factors may take any positive real values and need not be normalized, hence necessitating Z to normalize the product.

We define a *Markov network* as a pair (G, θ), where G is an undirected graph satisfying the Markov property and $\theta = (\theta_{C_1}, \ldots, \theta_{C_m}; Z)$ are the parameters.

2.1.3 Decomposable models

As a final model class we introduce the family of *decomposable graphical models* [68], which are exactly those models that can be represented as both a Bayesian network and a Markov network. Specifically, they are equivalent to Markov networks that are chordal and Bayesian networks where every two parents of every variable are joined by an edge. We will give a more formal definition via the commonly used junction tree representation, which will prove useful for learning these models from data.

We start by introducing the more general concept of tree decomposition, which will reoccur later in Chapter 5 in the context of variable elimination. To that end, let G be an undirected graph on the vertex set V and let $\mathcal{T}$

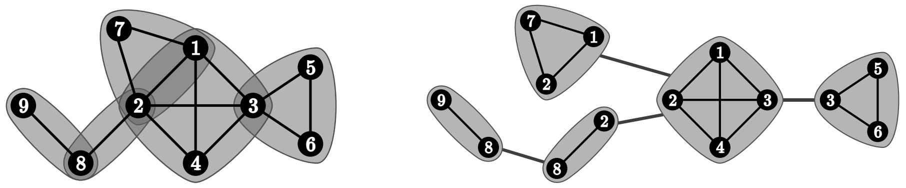


Figure 2.1: A decomposable graph (left) and one of its junction trees. The largest clique has 4 vertices and the treewidth of the graph is 3.

be a tree whose vertices $V_1, V_2, \ldots, V_m$ are subsets of V. We say that $\mathcal{T}$ is a *tree decomposition* of G if

1. every vertex $v \in V$ is contained in at least one V_i,
2. every edge $\{u, v\}$ of G is a subset of at least one V_i, and
3. $\mathcal{T}$ has the *running intersection property*: for every V_i, V_j we have that $V_i \cap V_j \subseteq V_k$ for every V_k on the unique path between V_i and V_j in $\mathcal{T}$.

We define the *width* of the decomposition as the size of the largest V_i, and the *treewidth* of G as the minimum width over all of its tree decompositions, minus 1.[1] Intuitively, the treewidth is a measure of how much G resembles a tree. For instance, every tree has a treewidth 1 while the complete graph on n vertices has treewidth $n - 1$.

We say that G is a *decomposable graph* if it has a tree decomposition $\mathcal{T}$ whose vertices are exactly the maximal cliques $C_1, C_2, \ldots, C_m$ of G. Such a $\mathcal{T}$ is called a *junction tree* and its every edge is associated with a *separator*, the intersection between the two cliques joined by the edge. A separator may be empty, and the same set may occur as a separator between multiple pairs of adjacent cliques. Although G may have several junction trees, each of them has the same separators, labeled $S_2, S_3, \ldots, S_m$. We denote the set of maximal cliques of G by $\mathcal{C}(G)$ and the multiset of separators by $\mathcal{S}(G)$. An example of a decomposable graph is depicted in Figure 2.1.

Now, if G is a decomposable graph on the variables V and has the global Markov property, we obtain the factorization (2.2). However, from the decomposable structure it follows that the factors may also be written as $\theta_i = \Pr(X_{C_i}) / \Pr(X_{S_i})$ for all $i = 1, \ldots, m$, where $\Pr(S_1) = 1$, and thus

[1]It is more common to include "minus 1" in the definition of the *width* rather than the treewidth. We deviate from this practice to avoid confusion between the width of the decomposition and the width of the graph, which is the size of the largest clique.

the joint probability factorizes as

$$\Pr(X_1, \dots, X_n) = \frac{\prod_{C \in \mathcal{C}(G)} \Pr(X_C)}{\prod_{S \in \mathcal{S}(G)} \Pr(X_S)} . \tag{2.3}$$

We thus define a *decomposable model* as a Markov network (G, θ) whose structure G is decomposable.

2.2 Structure learning

Our main interest is in learning graphical models automatically from data. We model a single *data point* as a vector $(x_1, x_2, \dots, x_n)$ representing a joint assignment of the n random variables. Let D denote the available data consisting of multiple data points, assumed to be independently drawn from a distribution of interest, called the *generating distribution*. Learning a graphical model from the data is typically separated into two phases, learning the structure and learning the parameters. If the structure is given, its parameters can be estimated independently for every induced factor, which is considered relatively straightforward [60]. From hereon we only consider the problem of learning the structure, which has turned out to be much more challenging both theoretically and computationally.

It is not obvious which structure within our model class of choice should be considered a best fit to the data. Although we would prefer a structure that asserts exactly the independencies of the generating distribution, such a structure might not exist in the model class. More importantly, it is not possible to infer the independencies with complete certainty from a finite amount of samples. To avoid overfitting to the limited data, we need some way to address the uncertainty about the distribution.

In Chapter 1 we outlined briefly two approaches to structure learning, called constraint-based learning and score-based learning. Most structure learning algorithms are based on one of these two approaches or a combination of them. We proceed to describe in greater detail the score-based approach, which will be our focus throughout the thesis.

2.2.1 Score-based learning

Suppose we have decided on a class of graphical models (either Bayesian networks or decomposable models) and wish to find a structure within that class to fit the data D. In score-based learning we assign to each potential structure G a non-negative real-valued *score* that measures how well G fits the data. We will denote the score by $s_D(G)$ or simply $s(G)$

assuming no ambiguity about D. While there are multiple ways to define what constitutes a good fit, a general approach is to prefer models that are more likely to have generated the data. To that end, scoring functions are typically based on the *likelihood* $\Pr(D \mid G, \theta)$, the probability of the data under the model (G, θ). Since we are only interested in evaluating the structure G, we need a way to define the likelihood without explicitly specifying the parameters θ. Commonly used scoring functions can be split into different families based on how they accomplish this.

We place focus on a category of functions known as the *Bayesian score.* The philosophy here is that, even before seeing the data, we should always have some prior beliefs regarding the structure and the parameters, ideally preferring simple models over complex ones. We model these beliefs as the *structure prior* $\Pr(G)$ and the *parameter prior* $\Pr(\theta \mid G)$, respectively. Given the data, we then measure the fit as the posterior probability of the structure, which according to Bayes' rule is

$$\Pr(G \mid D) = \frac{\Pr(D \mid G)\Pr(G)}{\Pr(D)}\,.$$

Since the normalizing constant $\Pr(D)$ does not depend on G, we typically ignore it and define the Bayesian score simply as $s(G) = \Pr(D \mid G)\Pr(G)$. The term $\Pr(D \mid G)$ is called the *marginal likelihood* and is obtained by integrating out the parameters,

$$\Pr(D \mid G) = \int_\theta \Pr(D \mid G, \theta)\Pr(\theta \mid G)\mathrm{d}\theta\,.$$

Under common assumptions about the parameter prior [46, 28] the integral can be computed exactly and the marginal likelihood factorizes along the structure G in a similar fashion as the probability in (2.1) and (2.3). This factorization is known as *modularity* of the likelihood.

Assuming a structure prior of similar form, modularity extends to $s(G)$ as well. Specifically, for Bayesian networks a *modular score* factorizes as

$$s(G) = \prod_{v \in V} s_v(G_v)\,,$$

where $s_v(G_v)$ is the *local score* associated with the variable v and its parents G_v. For decomposable models a modular score factorizes analogously as

$$s(G) = \frac{\prod_{C \in \mathcal{C}(G)} s_{\text{local}}(C)}{\prod_{S \in \mathcal{S}(G)} s_{\text{local}}(S)}\,, \tag{2.4}$$

where $s_{\text{local}}(U)$ is the local score defined for every subset $U \subseteq V$ of the variables. Modularity is an essential property for efficient structure learning as it allows evaluation of partial structures, where the presence or absence of particular edges has not yet been decided. This enables various techniques of combinatorial optimization that construct their solutions incrementally and avoid exhaustive search in the space of all structures. Modular scoring functions are sometimes alternatively called *decomposable*, though we will reserve this term for decomposable models and related concepts.

To obtain modularity, we had to assume a conforming structure prior. As it turns out, the parameter prior has a greater role in penalizing for excess complexity [60, Ch. 18]. Hence, a common and natural choice of a modular structure prior is the uniform prior, which does not favor any structure over another. For different choices of the parameter prior, this leads to scores such as K2 [23] and those in the BDe family [46] such as BDeu [17].

Other important scoring functions include the family of *maximum likelihood score*, which is based on the probability of the data when the parameters are fitted to the structure with maximum likelihood estimation. Since such a definition always prefers complete graphs with no independencies, the score is augmented with an information-theoretic penalization term, leading to scores such as AIC and BIC/MDL [13, 66, 106, 62]. A more recently proposed fNML score [100] is based on a modular derivative of the so-called normalized maximum likelihood criterion.

2.2.2 Computational problems

We proceed to describe the computational problems of structure learning. The input to these problems is assumed to be a modular Bayesian scoring function s, given as precomputed local scores.[2] The score defines a posterior distribution over all possible structures, which we wish summarize in some useful manner.

One common way to summarize the posterior distribution is to find a *maximum a posteriori* structure, a single graph that maximizes the score. We define this task formally as computing

$$\max_G s(G)\,,$$

where G runs over all structures. We call this the *optimization* variant of structure learning. Typically, a structure than attains the maximum score

[2]In practice, the local scores are usually evaluated and given in logarithmic form. This is done for the sake of numerical stability, as the probabilities are typically very small.

is also found in the process; there might be more than one such structure and any one of them can be chosen. Occasionally, we wish to augment the optimization problem with additional constraints that limit the complexity of the structure. For Bayesian networks, a natural constraint is to require that no variable may have more than k parents, where k is a small constant. For decomposable models one can similarly restrict the *width*, that is, the maximum size of every clique in the graph. In particular, such restrictions are often required to reduce the number of input scores to a manageable level. For Bayesian networks the input may also be reduced by pruning out any parent set that has a subset with a higher score, as any network containing such a parent set can be improved by substituting the subset instead [109, 29]. By contrast, for decomposable models such pruning rules appear not applicable. We note that the optimization problem as well as the generalizations are well-defined for all scoring functions, not only those based on Bayesian posterior probability.

While the graphical models we consider cannot express all sets of independencies exactly, most commonly used scoring functions guarantee that, as the amount of data grows, an optimal structure in the limit will be the simplest one that can represent the generating distribution [60, Ch. 18]. For small amounts of data, however, an optimal structure is likely to be heavily overfitted and may generalize poorly for new observations. In such cases, it can be preferable to learn multiple good structures and use them for model averaging.

Besides finding one or multiple representative graphs, one can learn about the structure by computing some interesting marginal of the posterior distribution. In the most general setting, we wish to compute the *posterior expectation*

$$\mathrm{E}[f \mid D] = \sum_{G} s(G)\, f(G) \tag{2.5}$$

for some arbitrary real-valued function f of the structure. This is known as *Bayesian averaging* [60, Ch. 18], and a typical case of it is the computation of posterior probabilities of graph features. For instance, if f is an indicator function for the presence of a particular edge in the graph, then the expectation (2.5) is the (unnormalized) posterior probability for the edge being present. The normalizing constant is also obtained as a special case of (2.5), by setting $f(G) = 1$ for all G.

Computing (2.5) within reasonable resources typically requires that the function f have a similar modular structure as the scoring function. When f does not have the desired modular form but one has access to *samples* $G^{(1)}, G^{(2)}, \ldots, G^{(T)}$ from the posterior distribution, it is possible to compute a Monte Carlo [94] estimate

$$\frac{1}{T}\sum_{i=1}^{T} f(G^{(i)})\,. \tag{2.6}$$

This estimate is unbiased and by the law of large numbers concentrates around the true expectation $\mathrm{E}[f \mid D]$ as T grows. Obtaining independent samples from the posterior is in general a nontrivial problem; however, an exact algorithm for computing posterior expectations can often be turned into an algorithm for producing such samples, as we will see later.

In some cases it turns out to be more efficient to compute a somewhat biased version of either (2.5) or (2.6). For example, particular MCMC schemes [37, 84] for sampling Bayesian networks attain faster mixing by first sampling a linear order on the variables and then sampling a network (DAG) for which the order is a topological sort. While this expedites the sampling, each DAG G will be sampled with a probability proportional to $s(G)\sigma(G)$, where $\sigma(G)$ denotes the number of topological sorts of G. Since the number of topological sorts may greatly vary between different DAGs, this introduces a bias to the Monte Carlo estimate. Such bias is not necessarily undesired, as it can often be viewed simply as using a different kind of (non-modular) structure prior. In general, however, we may want to correct the bias in order to employ a broader class of priors such as the natural uniform prior. One way to achieve this is to compute the self-normalized importance sampling estimate

$$\sum_{i=1}^{T} w_i\, f(G^{(i)}) \Big/ \sum_{i=1}^{T} w_i\,, \tag{2.7}$$

where the *importance weight* w_i counterweights the bias in each sample [94]. In this case, we would set $w_i = \sigma(G^{(i)})^{-1}$, in which case the estimate (2.7) is unbiased and concentrates around $\mathrm{E}[f \mid D]$.

In this particular example computing the importance weights is a hard problem, which we will revisit in Chapter 5. A similar usage of importance sampling also appears in Chapter 3, where the bias arises from the junction tree representation of decomposable models and is easier to counter.

2.2.3 Exact algorithms for structure learning

The optimization variant of structure learning is NP-hard for both general Bayesian networks [19, 21] and decomposable models [104], even assuming a modular scoring function. Regardless, several algorithms are guaranteed to find an optimal solution without resorting to exhaustive search. For Bayesian networks the first *exact* algorithms of this kind utilized Bellman–Held–Karp [5, 47] style dynamic programming over variable subsets, solving the problem in $O(2^n n^2)$ time and $O(2^n n)$ space for n variables [85, 58, 101, 99]. Later work has built upon the dynamic programming approach by presenting ways to parallelize the computation, reduce space complexity in exchange for increased running time, or incorporate prior knowledge on variable precedence [89, 87, 107]. Hybrid algorithms [93, 59] reduce the search space of the dynamic programming with additional constraints, which can first be learned with non-score-based methods.

While no algorithm is known to solve the problem faster in the worst case, various adaptive search techniques can exploit hidden structure in the input scores and achieve a better performance on many problem instances. One such approach expedites the dynamic programming by reformulating it as a shortest-path problem in the lattice of subsets and employing the A* algorithm with various best-first heuristics [121, 120, 35]. Another algorithm employs branch-and-bound search in a space of potentially cyclic graphs, considering all possible ways to break cycles to reach DAGs that are viable solutions [29]. On yet another front, structure learning has been formulated as a constraint optimization problem such as integer linear programming (ILP) [50, 26, 27, 4, 97] or constraint programming (CP) [112], allowing one to apply state-of-the-art solvers for these problems. Optimal networks of *bounded treewidth* can also be found by dynamic programming [61], by ILP based methods [86], or by casting into maximum satisfiability [6].

The feature expectation problem for Bayesian networks can likewise be solved by dynamic programming. Specifically, the posterior probability of an arbitrary subnetwork can be computed in $O(2^n n^2)$ time and space, assuming a so-called order-modular structure prior [58, 57]. Here, too, space usage can be reduced at the expense of time [88], and for a modular structure prior the computation can be carried out in $O(3^n)$ time [111].

For decomposable models both optimization and expectation appear somewhat harder, having no non-trivial worst-case bounds prior to the work presented in this thesis. Earlier work has, however, presented exact algorithms for the optimization variant by casting into a variety of constraint languages, including maximum satisfiability, satisfiability modulo theories, and answer set programming [25]. The ILP approach has also been applied to learning decomposable models, by adapting the algorithm designed for Bayesian networks [27] and, more recently, by employing different types of constraints altogether [105].

Chapter 3

Bayesian learning in decomposable models

We begin the contributions of this thesis by addressing the structure learning problem in decomposable models. We present a system of recurrence relations, which, via dynamic programming, enables computation of both the optimization variant and posterior expectations in exponential time with respect to the number of variables. We also extend these algorithms to a sampling scheme, which enables Monte Carlo estimation of expectations that the algorithms cannot compute exactly.

This chapter is based on Papers I and II.

3.1 Decomposable functions

As elaborated in Chapter 2, efficient structure learning requires that the scoring criteria and features have a certain modular structure. Specifically, consider a real-valued function φ defined on all decomposable graphs on the random variables. We say that φ is a *decomposable function* if

$$\varphi(G) = \frac{\prod_{C \in \mathcal{C}(G)} \varphi_{\mathrm{c}}(C)}{\prod_{S \in \mathcal{S}(G)} \varphi_{\mathrm{s}}(S)} \tag{3.1}$$

for some φ_{c} and φ_{s} called the *local components* of φ. It is immediate that a modular score (2.4) is a special case of a decomposable function where the components φ_{c} and φ_{s} coincide.

Certain natural features of decomposable graphs also turn out to be decomposable. For instance, let w be a number and let φ^w be the decomposable function defined by the local components

$$\varphi_{\mathrm{c}}^w(X) = [\,|X| \le w\,] \quad \text{and} \quad \varphi_{\mathrm{s}}(X) = 1\,, \quad \text{for all } X \subseteq V\,,$$

where the Iverson bracket $[Q]$ denotes 1 if Q is true and 0 otherwise. Thus φ is an indicator function for the property of having width at most w, which is therefore a *decomposable feature.* As another example, consider an edge $\{u, v\} \subseteq V$ and let φ be the decomposable function defined by

$$\varphi_{\mathrm{c}}(X) = [\{u, v\} \not\subseteq X] \quad \text{and} \quad \varphi_{\mathrm{s}}(X) = 1\,, \quad \text{for all } X \subseteq V\,.$$

Then φ is an indicator function for the *absence* of the edge $\{u, v\}$. In particular, any constant function is decomposable, and the product of two decomposable functions is also decomposable.

In this light, the structure learning tasks defined in Chapter 2 can be stated in terms of maximizing or summing a decomposable function over all structures. For instance, to solve the optimization problem for bounded width w we would compute $\max_G \varphi(G)$ for $\varphi = s\varphi^w$, where s is any modular score. Likewise, to solve the feature expectation problem, we would compute $\sum_G \varphi(G)$ for $\varphi = sf$, where s is a Bayesian score and f is a decomposable feature. We will next address the maximization and summation problems separately for an arbitrary decomposable function, giving a similar dynamic programming approach for both of them.

3.2 Maximization over decomposable graphs

Let φ be the decomposable function of interest and consider the problem of computing $\max_G \varphi(G)$. To avoid an exhaustive enumeration of all graphs, we formulate the maximization as a recurrence relation that exploits the common structure present in G and φ. Specifically, recall that a graph is decomposable if and only if there is a junction tree on its maximal cliques. Consider an arbitrary junction tree $\mathcal{T}$ and denote by $G(\mathcal{T})$ the unique decomposable graph whose maximal cliques are the vertices of $\mathcal{T}$. We may treat $\mathcal{T}$ as a directed tree by picking an arbitrary maximal clique C as its root. Let $C_1, \ldots, C_k$ be the children of C in $\mathcal{T}$ and denote by $\mathcal{T}_1, \ldots, \mathcal{T}_k$ the respective subtrees rooted at the children. Then, writing $\varphi(\mathcal{T}) = \varphi(G(\mathcal{T}))$ for short and recalling that separators are the intersections between adjacent cliques, we may express (3.1) recursively as

$$\varphi(\mathcal{T}) = \varphi_{\mathrm{c}}(C) \prod_{i=1}^{k} \frac{\varphi(\mathcal{T}_i)}{\varphi_{\mathrm{s}}(C \cap C_i)}\,. \tag{3.2}$$

Computing $\max_G \varphi(G)$ is equivalent to computing $\max_{\mathcal{T}} \varphi(\mathcal{T})$, where $\mathcal{T}$ runs over all rooted junction trees. To see how (3.2) yields a recurrence for this problem, consider a recursive procedure for constructing a junction tree

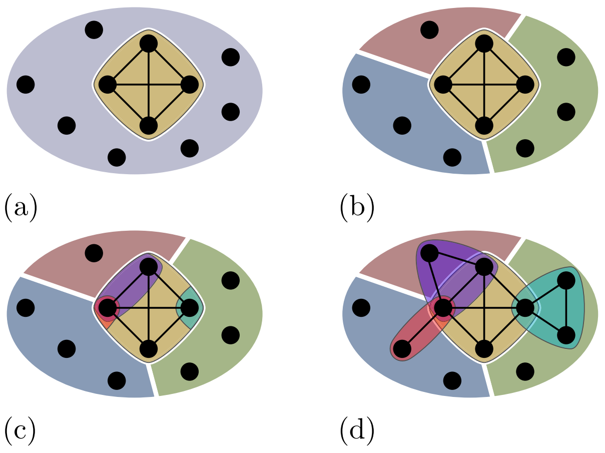


Figure 3.1: Recursive construction of a junction tree. We begin by choosing an arbitrary root clique (a). Next, we choose a partition of the vertices outside the clique (b). We then choose a separator within the root clique for each set in the partition (c). Finally, we apply the first step recursively to every subtree, choosing a root clique for each of them (d).

on the vertex set V, illustrated in Figure 3.1. First, the procedure chooses a set of vertices $C \subseteq V$ to be the root clique. Next, the procedure chooses a partition of remaining vertices, denoted $\{R_1, R_2, \dots, R_k\} \sqsubset V \setminus C$, with the intent that each set R_i in the partition will contain a single subtree rooted at C. For each R_i, independently, the procedure chooses a separator $S_i \subset C$, and then the entire procedure is applied recursively for each subtree, with the constraint that every root clique C_i must satisfy $S_i \subset C_i$.

Clearly, every rooted junction tree on the variables can be constructed in this manner, by choosing the cliques, partitions, and separators recursively. Similarly, by *maximizing* the score over all such choices we maximize it over all rooted junction trees.

Formally, for all $S \subset V$ and $\varnothing \neq R \subseteq V \setminus S$, let $f(S, R) = \max_{\mathcal{T}} \varphi(\mathcal{T})$, where $\mathcal{T}$ runs over every rooted junction tree whose vertex set is $S \cup R$ and whose root clique is a proper superset of S. Then, we have that

$$f(S,R) = \max_{\substack{S \subset C \subseteq S \cup R \\ \{R_1,\dots,R_k\} \sqsubset R \setminus C \\ S_1,\dots,S_k \subset C}} \varphi_{\mathrm{c}}(C) \prod_{i=1}^{k} \frac{f(S_i, R_i)}{\varphi_{\mathrm{s}}(S_i)}\,. \tag{3.3}$$

In particular, $f(\varnothing, V)$ now equals the maximum score over all rooted junction trees on the vertex set V.

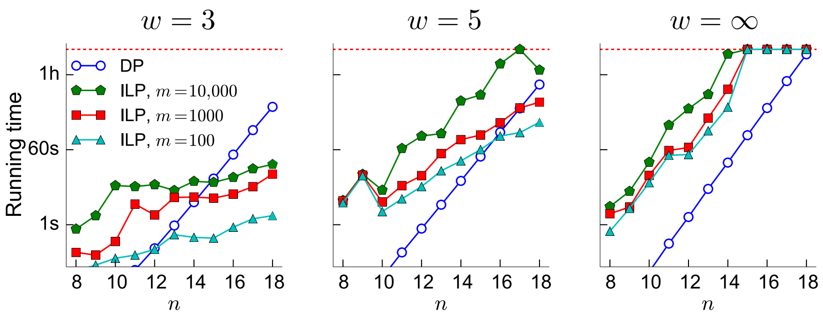


Figure 3.2: The running time of the DP and ILP algorithms on synthetic data, using the BDeu score. Each data point represents the median of the running times on 15 independently generated problems with the same number of variables n. For ILP, the performance is measured separately for a varying number of samples m in the data. Both algorithms are evaluated for restricted width $w \in \{3, 5\}$ as well as unbounded width ($w = \infty$).

With some basic manipulation the maximization in (3.3) can be split into three parts, allowing us to write it as a system of simpler recurrences

$$f(S,R) = \max_{S \subset C \subseteq S \cup R} \varphi_{\mathrm{c}}(C)\, g(C, R \setminus C)\,, \tag{3.4}$$

$$g(C,U) = \max_{\min U \in R \subseteq U} h(C,R)\, g(C, U \setminus R)\,, \tag{3.5}$$

$$h(C,R) = \max_{S \subset C} f(S,R)/\varphi_{\mathrm{s}}(S)\,, \tag{3.6}$$

terminating at the base case $g(C, \varnothing) = 1$. Each of these recurrences is defined for all disjoint pairs of subsets of V such that C and R are nonempty. Hence, a straightforward dynamic programming algorithm tabulates all values of f, g, and h in $O(4^n)$ time and $O(3^n)$ space. A graph G that maximizes $\varphi(G)$ can be found afterwards by a simple backtracking into the dynamic programming tables.

Somewhat better bounds on time and space can be achieved when certain easily characterized portions of φ_{c} and φ_{s} are zero. For instance, when searching for optimal graphs of maximum width $w \leq n$, we have $\varphi_{\mathrm{c}}(C) = 0$ for every $C \subseteq V$ such that $|C| > w$. Hence, we may evaluate recurrence (3.4) with the additional constraint $|C| \leq w$ without changing the result. Omitting larger cliques from consideration improves the performance, requiring $O(\sum_{i=0}^{w} \binom{n}{i} 3^{n-i})$ time and $O(\sum_{i=0}^{w} \binom{n}{i} 2^{n-i})$ space instead. When $w \leq n/4$, the former bound can be replaced by $O(w \binom{n}{w} 3^{n-w})$. The details of this derivation and the analysis of time and space are included in Paper I.

Table 3.1: Benchmark instances from machine learning repositories [73, 44] with different numbers of variables (n) and samples (m).

Dataset		n	m
Tic-tac-toe	X	10	958
Poker	P	11	10000
Bridges	B	12	108
Flare	F	13	1066
Zoo	Z	17	101
Voting	V	17	435
Tumor	T	18	339
Lymph	L	19	148
Hypothyroid		22	3772
Mushroom		22	8124

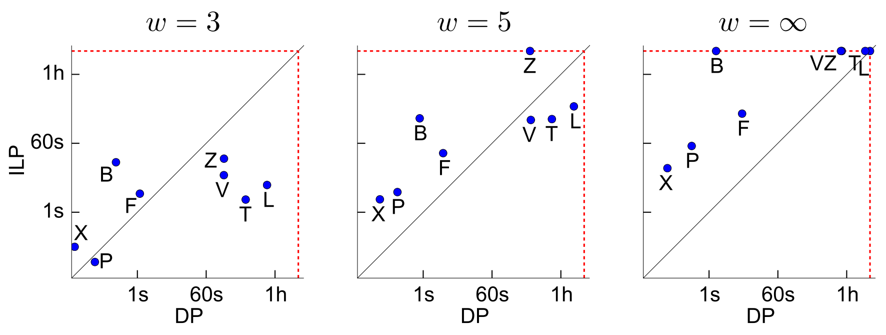


Figure 3.3: The running time of the DP and ILP algorithms on the benchmark instances.

Paper I evaluates the performance of the dynamic programming (DP) algorithm against an integer linear programming (ILP) based approach [27], which likewise guarantees the optimality of its output but has an unknown worst-case complexity. The comparison is shown in Figure 3.2 for data sampled from synthetic networks and in Figure 3.3 for benchmark datasets summarized in Table 3.1. The results suggest that the DP algorithm performs better for very small instances ($n < 10$) as well as for moderate and large widths w, while the ILP algorithm can be faster on larger instances as long as w is small. The ILP approach is also sensitive to the input scores, performing worse as the sample size m increases, whereas the running time of DP is strictly a function of n and w. Two larger problems ($n = 22$) were given a one-week timeout. The DP algorithm solved both within 33 hours for $w = 3$ and 74 hours for $w = 4$. The ILP algorithm solved Hypothyroid up to $w = 6$ within 24 hours and Mushroom up to $w = 3$.

3.3 Summation over decomposable graphs

Consider now the problem of computing $\sum_G \varphi(G)$ over all decomposable graphs G. It is easy to see that one can adapt the dynamic programming algorithm to compute the respective summation over junction trees, by simply replacing maximization with summation in recurrences (3.4), (3.5), and (3.6). Unlike with maximization, however, $f(\varnothing, V)$ will not equal the desired summation over all decomposable graphs due to the one-to-many relationship between graphs and junction trees. Letting $\tau(G)$ denote the number of rooted junction trees of G, we instead have that

$$f(\varnothing, V) = \sum_{\mathcal{T}} \varphi(G(\mathcal{T})) = \sum_{G} \varphi(G)\tau(G)\,,$$

where $\mathcal{T}$ runs over rooted junction trees and G runs over decomposable graphs. Intuitively, the computed sum will be biased towards graphs with many junction trees. As noted in Chapter 2, such bias can be viewed as using a special kind of structure prior that is modular with respect to rooted junction trees rather than decomposable graphs. For now, we will thus accept the bias as a natural effect arising from the junction tree representation. At the end of this chapter we consider countering the bias and estimating the preferred sum $\sum_G \varphi(G)$ via importance sampling.

While summation still admits the straightforward evaluation of the recurrences, it also enables more sophisticated techniques. Specifically, with an extension of the so-called *fast zeta transform* [118, 56, 12] and *fast subset convolution* [9] the recurrences can be computed asymptotically faster, in $O(3^n n^3)$ time and $O(3^n n)$ space. In particular, the posterior probability that k specified edges appear in the graph can be computed, via the principle of inclusion and exclusion, in $O(2^k 3^n n^3)$ time and $O(n3^n)$ space. The proofs of these results are presented in detail in Paper II.

3.4 Sampling schemes

We next consider the problem of sampling decomposable graphs from the posterior, which is useful for performing (approximate) Bayesian model averaging as well as estimating posterior expectations for non-decomposable features. We can turn the dynamic programming algorithm for summation into a sampling algorithm by first computing the dynamic programming tables for a Bayesian score φ and then employing stochastic backtracking.

To exemplify, consider the summation variant of recurrence (3.4),

$$f(S, R) = \sum_{S \subset C \subseteq S \cup R} \varphi_{\mathrm{c}}(C)\, g(C, R \setminus C)\,.$$

The recurrence gives the marginal probability $f(S, R)$ of a particular subset of rooted junction trees defined by S and R. Each term $\varphi_{\mathrm{c}}(C)\, g(C, R \setminus C)$ is the conditional probability that a tree within this subset has C as its root clique. Similar observations can be made for the partition sets R and the separators S in recurrences (3.5) and (3.6). A natural backtracking procedure starting at $f(\varnothing, V)$ first draws a clique from the marginal distribution $\Pr(C) \propto \varphi_{\mathrm{c}}(C)\, g(C, R \setminus C)$ and then recurses on g to draw the other cliques, partitions, and separators in a similar fashion. Note that the bias of the summation transfers to sampling: the graph obtained in this manner will be drawn from the distribution proportional to $\varphi\tau$ instead of φ.

The efficiency of this backtracking procedure depends on how fast we can sample each C, R, and S. We consider two distinct sampling methods, which we will call parameterized sampling and adaptive sampling.

3.4.1 Parameterized sampling

Drawing each individual set involves sampling from a discrete distribution over s elements, where s can be at most 2^n. A naive method draws a number r from the continuous uniform distribution on $[0, r)$, then iterates over all elements in a predefined order until the cumulative probability exceeds r, thus using $O(s)$ time and $O(1)$ space. The *alias method* [114] by contrast requires $O(s)$ time and space to first construct a data structure, which thereafter allows one to draw each sample in $O(1)$ time.

By combining the naive method and the alias method we can obtain a tunable tradeoff between time and space. Specifically, let $b \in \{0, ..., n\}$ be a tradeoff parameter that is chosen beforehand. In the preprocessing step we first divide the s elements into bins of size 2^b (the last bin can be smaller). For each bin, we compute the sum of the probabilities of its elements, then construct the alias data structure for sampling a bin according to its total probability. This construction requires $O(2^b)$ time and $O(s/2^b)$ space. After the preprocessing, an element can be drawn in two phases: first a bin is selected with the alias method in $O(1)$ time and then an element within the bin is selected with the naive method in $O(2^b)$ time.

We now select a fixed b and apply this method to each (nonterminating) $f(S, R)$, $g(C, U)$, and $h(C, R)$. The analysis given in Paper II shows that we require $O(4^n)$ time and $O(4^n/2^b)$ extra space for preprocessing in total, and can then draw T independent samples in $O(T \cdot 2^b(1 + n - b))$ time.

In particular, setting $b = 0$ allows us to draw each sample in linear time but using $O(4^n)$ extra space. The other extreme, $b = n$, requires $O(2^n)$ time per sample but allows us to omit the alias data structures altogether, thus requiring no extra space. Notably, by setting $b \simeq n \log_2(4/3) \simeq 0.42n$ we can draw $O(4^n/2^{n \log_2(4/3)} n) = O(2^n/n)$ samples for "free", without exceeding the asymptotical $O(4^n)$ time or $O(3^n)$ space already required by the dynamic programming phase.

3.4.2 Adaptive sampling

For typical real-world data, the posterior distribution tends to be highly skewed, with most of the probability mass concentrated on a small set of graphs. In practice it may thus be wasteful to do the preprocessing for all indexing pairs for f, g, and h, as many of these pairs are visited only rarely if at all during the sampling procedure. As an alternative, we consider an adaptive approach that does no preprocessing but instead preemptively draws multiple samples at once on those indices that are visited more often, caching them for later visits. Specifically, consider the following procedure. On the first visit to any $f(S, R)$, $g(C, U)$, or $h(C, R)$, draw and consume one set from the corresponding distribution using the naive sampling method. On any subsequent visit: If there are cached samples left from previous visits, consume the first one of them. Otherwise, use the alias method to draw twice as many sets as the last time on the same index, consume the first one of those sets and cache the rest.

By doubling the number of samples every time we effectively amortize the linear time required to construct the alias data structure. We can show that, after T sampled graphs, adaptive sampling consumes at most $O(nT)$ extra space. As a special case, 3^n samples can be drawn in $O(n4^n)$ time and $O(n3^n)$ space. The proofs of these results can be found in Paper II.

3.5 Monte Carlo estimation

The sampling schemes yield an immediate tool for computing the Monte Carlo estimate (2.6) for an arbitrary posterior expectation, so long as we accept the non-modular prior induced by the summation. The importance sampling estimate (2.7) allows us to target a broader class of priors such as the natural uniform prior over decomposable graphs, provided that we can efficiently compute the appropriate importance weights. To target an arbitrary modular prior, we apply the summation algorithm to a Bayesian score φ of choice and define the importance weights $w_i = \tau(G)^{-1}$ for each graph G_i sampled. The number $\tau(G)$ of rooted junction trees is obtained

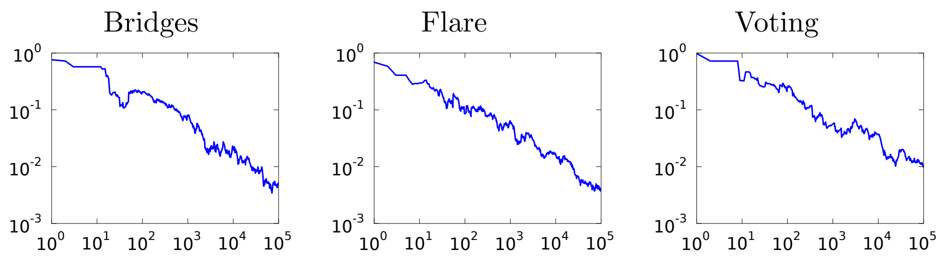


Figure 3.4: The maximum error in the estimated edge probabilities over all edges (y-axis) as the number of samples (x-axis) grows, for three benchmark datasets. As computing the true edge probabilities is infeasible, the error is computed by comparison to an estimate obtained with 10^6 samples.

as a product of the number of (undirected) junction trees and the number of maximal cliques of G. Counting the cliques is easy as they are readily available in the junction tree representation of G_i. For counting junction trees the fastest known algorithm [110] has a guaranteed $O(n^2)$ running time, though its performance can be much better in practice.

Experimental results presented in Paper II demonstrate that the estimate has a steady rate of convergence in practice. Figure 3.4 shows these results for edge posterior probabilities and the uniform prior.

Chapter 4

Algorithm selection for learning Bayesian networks

Chapter 3 studied the structure learning problem in decomposable models, where we have only very recently seen attempts at solving the problem to optimum. By contrast, for general Bayesian networks the problem of optimal structure learning has attained considerable interest in the past decade. As elaborated in Chapter 2, several *exact* algorithms have been proposed that are guaranteed to find a network structure with the highest score. As state-of-the-art algorithms rely on inherently different techniques, their relative performance typically varies greatly depending on the given problem instance. Indeed, empirical analysis has demonstrated that no single algorithm currently dominates the others in terms of speed: rather, different algorithms perform well on different types of instances.

In this chapter we address the problem of algorithm selection [96, 63] for structure learning in Bayesian networks: Given a previously unseen problem instance, which exact algorithm should we run to minimize the expected running time? To tackle this problem, we learn so-called empirical hardness models [70, 71] for predicting the running time of an algorithm on a given problem instance, based on efficiently computable *features* of the instances and a large-scale empirical evaluation of the algorithms. Such models in turn enable the construction of *algorithm portfolios* [40, 49] that aim to run the fastest algorithm on any given instance, based on the predictions given by the models. This approach has proven successful for a variety of other NP-hard search problems [64] such as Boolean satisfiability [117], answer set programming [48], and the travelling salesman problem [65].

This chapter is based primarily on Paper IV, which extends the preliminary study presented in Paper III.

4.1 Overview

The optimization problem for Bayesian networks asks for a DAG G that maximizes a modular score $s(G) = \prod_{v=1}^{n} s_v(G_v)$, where n is the number of variables and $s_v(G_v)$ is a local score associated with variable v and its set of parents G_v. A single *instance* of the problem is assumed to consist of the precomputed local scores. As noted in Chapter 2, the size of an instance may be reduced by enforcing a small upper bound on the maximum number of parents per variable or by pruning out provably nonoptimal parent sets. We call every remaining set a *candidate parent set* (CPS) and denote by m the total number of CPSs of all variables.

Papers III and IV study the behavior of four state-of-the-art exact algorithms for the optimization problem: the URLearning solver [120], based on heuristic search (A*); the GOBNILP solver [27], based on integer linear programming (ILP); the CPBayes solver [112], based on constraint programming (CP); and a branch-and-bound search (BB) in the space of cyclic graphs [29]. Out of these, the BB algorithm was found by an empirical evaluation to be always dominated by the ILP approach. The remaining three, on the other hand, appear highly orthogonal in terms of performance. As shown in Figure 4.1, not only is no single algorithm consistently faster than the other two, the running times between algorithms can vary by several orders of magnitude on the same problem instance. To improve upon the state-of-the-art, an ideal approach would thus be to consult an oracle that knows (beforehand) the fastest algorithm for any given instance. Our goal is to approximate a perfect oracle by using machine learning methods to predict the running time of an algorithm A on a given instance. Specifically, we consider the following supervised learning task:

Input: A collection of training instances, $x_1, x_2, \ldots, x_N$ represented as features vectors, and the respective running times $y_1, y_2, \ldots, y_N$ of algorithm A on the instances.

Task: Learn an efficiently computable function f_A that, given a new problem instance x, yields a prediction $\hat{y} = f_A(x)$ of the true running time y of algorithm A on x, so as to minimize the average (relative) prediction error.

4.2 Features

The main challenge in constructing a successful predictor is to identify features of the problem instances that are efficient to compute yet informative

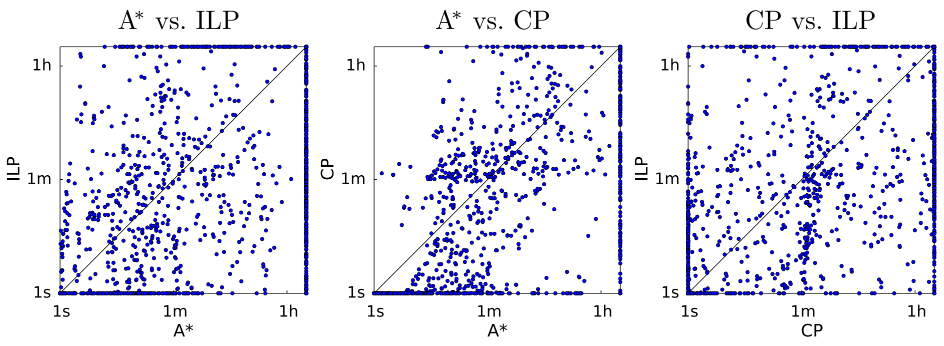


Figure 4.1: Running time comparison of three exact algorithms, based on A* search, integer linear programming (ILP), and constraint programming (CP). Each point represents a single problem instance derived from a collection of benchmark and synthetic datasets. Running times below 1 or above 7200 seconds are truncated to 1 and 7200, respectively.

about the running times of algorithms. We identify and study a total of 86 potentially useful features, divided into four categories.

For the first category, **Basic**, we consider two natural measures of instance size, the number of variables n and the number of CPSs m. Specifically, we include n and the *mean* number of CPS, m/n, which can be viewed as a measure of density of the instance. Various other statistics can be extracted directly from the number and size of CPSs, which we include in the second category, **Basic extended**.

Relaxing the problem can yield useful upper bounds on the score of an optimal network. For instance, the *simple upper bound* is obtained by letting every variable choose its best parent set independently of other variables. Although this usually results in a graph that is cyclic and thus not a valid solution, the structural properties of the graph can be useful predictors of instance hardness. We include such features in our third category, **Upper bounding**, including statistics of vertex degrees and connected components. We also consider a more sophisticated upper bound obtained with so-called *pattern databases* [119], which find optimal structures within small groups of variables but allow cycles between the groups.

Probing refers to running an algorithm for a few seconds and collecting statistics of its behavior. In particular, all of the three exact algorithms have anytime characteristics: when stopped, they output the best graph found so far together with a guaranteed error bound for its optimality. We probe with all three algorithms as well as greedy search and include

Table 4.1: Features extracted for every problem instance. Here, sd stands for standard deviation and NTSSC stands for non-trivial strongly connected component. The upper bounding features are computed both for the simple and the pattern database upper bound. The probing features are computed for four different probing strategies: greedy, A*, ILP, and CP.

Basic	**Basic extended**
– Number of variables, n	– Number of CPSs: max, sum, sd
– Mean number of CPSs, m/n	– CPS sizes: max, mean, sd
Upper bounding ($\times 2$)	**Probing** ($\times 4$)
– In-degree: max, mean, sd	– In-degree: max, mean, sd
– Out-degree: max, mean, sd	– Out-degree: max, mean, sd
– Total degree: max, mean, sd	– Total degree: max, mean, sd
– Number of root variables	– Number of root variables
– Number of leaf variables	– Number of leaf variables
– Number of NTSCCs	– Error bound
– Size of NTSCCs: max, mean, sd	

the resulting error bounds and graph properties in our fourth category, **Probing**. For purpose of comparison, we will denote by **All** the category encompassing all features of the four categories, summarized in Table 4.1.

4.3 Experiment setup

Our approach relies on carrying out an empirical evaluation of the three algorithms on a collection of typical instances. Specifically, we evaluate the following eight parameterizations: For A* we consider three variants, A*-ed3, A*-ec, and A*-comp, each of which uses a different heuristic. For ILP we consider four configurations, ilp-141, ilp-141-nc, ilp-162, and ilp-162-nc, based on versions 1.4.1 and 1.6.2, and additionally differing in how they solve sub-ILPs. Lastly, we consider cpbayes, the default configuration of the CP algorithm, which exposes no parameters to control its behavior.

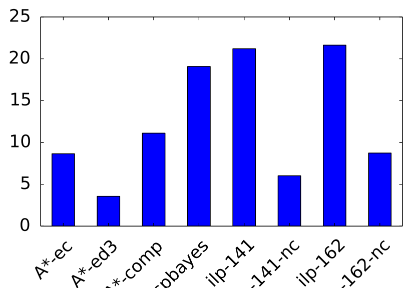
We evaluate each configuration on a large collection of instances, divided into three categories based on the origin of the underlying dataset. The first category, Real, contains instances based on datasets that originate from real-world applications, obtained from machine learning repositories [73, 44]. The second category, Sampled, contains instances sampled from handmade Bayesian networks that are widely used for evaluating indi-

Table 4.2: Number of datasets, instances derived from the datasets, and instances used in training and testing the models.

Category	Datasets	All Instances	Training & Testing
Real	39	637	486
Sampled	19	317	283
Synthetic	477	477	410

vidual solvers. The third category, Synthetic, contains instances sampled from purely synthetic Bayesian networks, generated by a randomized procedure. We translate the data into instances by applying a variety of scoring functions, including the Bayesian score BDeu and the penalized likelihood score BIC/MDL. We also consider various limits on the maximum of number of parents per variable. The total number of datasets and resulting instances is summarized in Table 4.2. For training and testing we omit all instances that are either very easy for all algorithms or not solved by any of them within the given time limit of 2 hours.

Figure 4.2 shows a comparison of algorithm performance in the empirical evaluation. Based on these results, we focus on a set of representative configurations, A*-comp, ilp-141, and cpbayes, as these variants have arguably the best overall performances within their respective families.



Algorithm	#solved	runtime (s)
ilp-141	1036	1,364,855
ilp-141-nc	1034	1,384,022
ilp-162	1029	1,453,932
ilp-162-nc	1026	1,494,879
cpbayes	896	2,423,547
A*-comp	768	3,152,809
A*-ec	519	4,866,797
A*-ed3	478	5,163,876

Figure 4.2: Left: The percentage of instances on which an algorithm was fastest, with ties broken at random. Right: The total number of instances solved and the total cumulative running time (failed runs count as 2 hours).

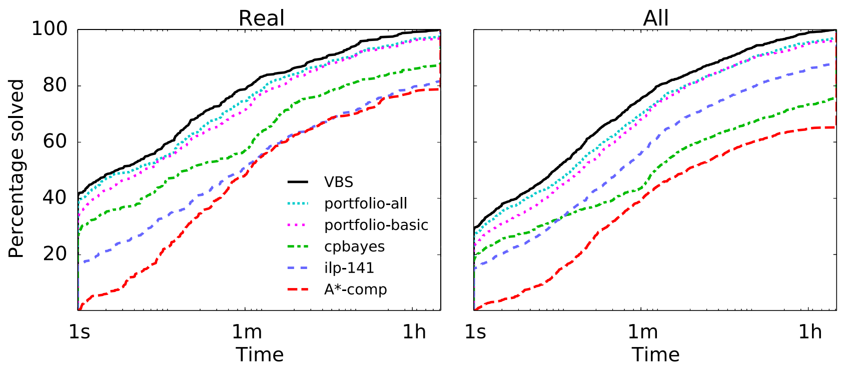


Figure 4.3: The proportion of instances solved by algorithms within a given amount of time, including the VBS and the two portfolios, for REAL instances and all instances.

4.4 Model training and evaluation

We use regression trees [14] to model the empirical hardness function f_A for every algorithm A. The preliminary study of Paper III also considers other models, such as reduced error pruning trees [95] and M5′ trees [116]. We train the models on a mix of all three categories of instances and use 10-fold cross-validation to evaluate them. Specifically, for a fixed algorithm A we partition the set of all instances into *folds* $F_1, \ldots, F_{10}$ evenly at random and define $f_A(x) = f_A^i(x)$, where i is such that $x \in F_i$ and f_A^i is a model that we train on the union of all 10 folds except F_i. We consider this procedure for various subsets of features and evaluate the resulting functions f_A in terms of how closely the predictions match the true running times and how well they translate into efficient algorithm portfolios.

4.4.1 Portfolio performance

A set of predictive models f_A immediately yields an algorithm portfolio that, given a new instance x, computes the predicted running time $f_A(x)$ for all algorithms A and then runs the algorithm with the lowest prediction. The performance of such a portfolio is commonly evaluated against the Single Best Solver, the algorithm with the lowest overall running time, and the Virtual Best Solver (VBS), a hypothetical portfolio that always runs the fastest algorithm and thus represents the best possible behavior a portfolio may theoretically achieve on a given set of algorithms.

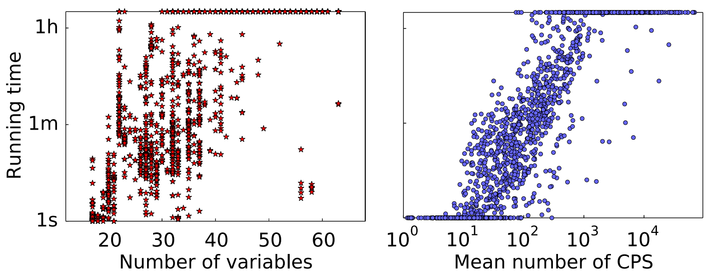


Figure 4.4: Correlation between the **Basic** features and the running times of A* (left) and ILP (right).

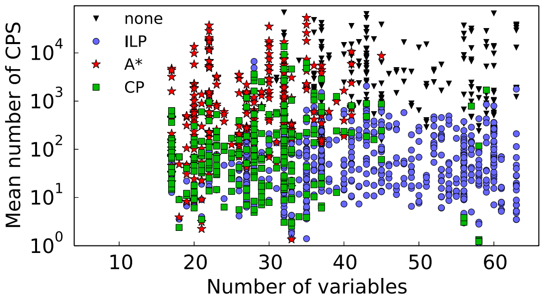


Figure 4.5: The **Basic** features plotted against each other and marked according to whether the fastest algorithm on the instance was from the A*, ILP, or CP family, or whether no algorithm solved the instance.

We find that even predictions obtained with very simple features are enough to construct a highly efficient portfolio. Figure 4.3 presents a comparison between two portfolios, portfolio-basic, which only uses the **Basic** features (n and m/n) to choose an algorithm, and portfolio-all, which uses all features of the four categories. While there remains some room for improvement compared to the VBS, portfolio-basic performs nearly as well as portfolio-all and significantly outperforms every individual algorithm.

We can gain insight into the success of the simple portfolio by studying the role of the two **Basic** features in determining the fastest algorithm. To that end, Figure 4.4 presents a correlation between the two features and running times of the A* and ILP algorithms. Figure 4.5 provides an alternative view, highlighting the advantages of different algorithms in the space spanned by these two features. Evidently, the A* algorithm is rather heavily limited by the number of variables but scales better for many CPSs.

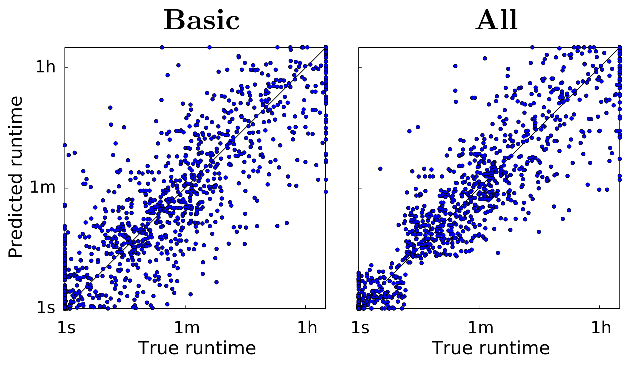


Figure 4.6: The actual runtimes of the ILP solver compared to the predicted runtimes when using the **Basic** (left) or **All** features.

ILP, on the other hand, cannot cope with a high number of CPSs but has no trouble with a large number of variables. The CP approach appears to excel on instances with a moderate number of both variables and CPSs, while A* or ILP outperforms it when either feature grows too high.

4.4.2 Prediction accuracy

We now consider the harder problem of obtaining accurate estimates for the running time of algorithms. Aside from selecting the best algorithm on a given instance, such estimates can be useful for job schedulers that need to determine the time required to complete a task.

While the more sophisticated features offered no notable improvement on portfolio performance, we find that they have a significant impact on the quality of predictions. Figure 4.6 illustrates this improvement for the ILP algorithm. On the left we see the predictions given by the model that only uses the **Basic** features. Even though these predictions are sufficient for good portfolio performance, they exhibit a considerable amount of error, often by several orders of magnitude, which makes them less useful for obtaining actual estimates of running time. The right side, on the other hand, shows the predictions based on **All** features, demonstrating a clear increase in accuracy. A similar improvement can be observed for the other algorithms as well. Table 4.3 summarizes these observations in terms of change in the approximation factor, defined as $\rho = \max\{\frac{a}{p}, \frac{p}{a}\}$, where a and p are the actual and predicted running times, respectively. Paper IV also presents a more fine grained analysis on the effect of incrementally adding various categories of features.

Table 4.3: The percentage of instances with an approximation factor within the given ranges of ρ, when predicting running times with either **Basic** or **All** features. Higher percentages with lower approximation values indicate more accurate predictions.

	A*-comp		**cpbayes**		**ilp-141**	
Range of ρ	**Basic**	**All**	**Basic**	**All**	**Basic**	**All**
< 2	48%	60%	49%	65%	52%	63%
$[2, 5)$	22%	17%	24%	18%	28%	26%
$[5, 10)$	10%	8%	10%	7%	10%	7%
> 10	21%	15%	18%	9%	10%	4%

4.4.3 Impact of features

Aside from practical applications, empirical hardness models can offer insight into what makes the problem hard or easy for a particular algorithm, potentially inspiring development of novel techniques that overcome the shortcomings of existing algorithms. Such knowledge can be extracted by studying how important certain features are for the predictive model.

Papers III and IV study the utility of the features assigned to them by the learning algorithms. While the utilities vary somewhat between different model classes, the **Basic** features are consistently identified as the most significant predictors of running time. Another unsurprising result is that the probing features are relevant for predicting the running times of the respective algorithms, though the probing features of CP in particular are also useful for predicting A*. These two algorithms directly utilize pattern database upper bounds in their search, and the respective lower bound features are identified as important for predicting both. Further analysis and discussion on the models and the relevance of features is presented in the papers.

Chapter 5

Counting linear extensions

In Chapter 2 we mentioned how certain MCMC schemes [37, 84] for sampling Bayesian networks attain fast mixing at the expense of drawing their samples from a biased distribution. Specifically, the probability of each DAG gets weighted according to the number of its topological sorts, and countering the bias via importance sampling thus requires one to compute this number for every sample.

Counting the topological sorts of a DAG is equivalent to counting the linear extensions of the corresponding partial order, a fundamental problem of order theory with applications in various areas such as sorting [92], sequence analysis [79], convex rank tests [82], preference reasoning [75], partial-order plans [83], and other algorithms for learning graphical models [115]. Unlike in Chapter 3, where computing the importance weights turned out to be easy, the problem of counting linear extensions is known to be #P-complete [15] and is thus unlikely to be tractable in the general case. The fastest known algorithm [74] for partial orders on n elements runs in $O(2^n n)$ time in the worst case. While polynomial time algorithms exist for various special cases [81, 3, 43, 36] such as polytrees and series-parallel orders, DAGs sampled from the posterior rarely fall into these categories. Randomized approximation schemes [33, 16] are likewise unsatisfactory as they tend to be impractically slow to attain desired error bounds.

In this chapter we consider exact counting of linear extensions of general partial orders. We present two algorithms, one based on recursive decomposition of the problem and another based on the method of variable elimination. Though neither algorithm avoids the exponential time in the worst case, they can be significantly faster when the partial orders are sparse, thus making them practical for typical Bayesian network structures. This chapter is based on Paper V.

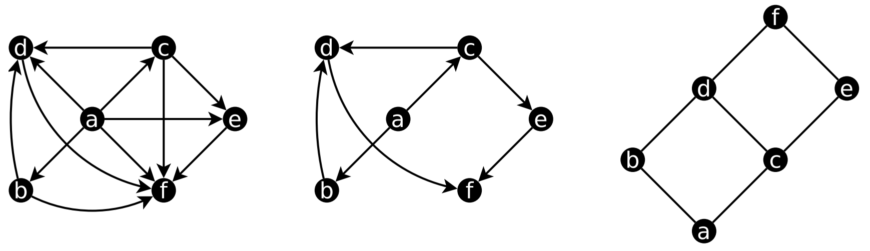


Figure 5.1: Three graph representations of a poset $(P, \leq_P)$. Left: a DAG with an arc $x \to y$ for all $x <_P y$. Middle: a DAG with an arc $x \to y$ for all $x \prec_P y$. Right: a Hasse diagram, with an undirected edge going upwards from x to y for all $x \prec_P y$.

5.1 Partial orders

Let P be a finite set associated with a binary relation $\leq_P$, whose elements $(x, y) \in \leq_P$ we will write simply $x \leq_P y$. We say that $\leq_P$ is a *partial order* on P if for all $x, y, z \in P$ the following three properties hold:

1. $x \leq_P x$ (reflexivity),
2. if $x \leq_P y$ and $y \leq_P x$ then $x = y$ (antisymmetry),
3. if $x \leq_P y$ and $y \leq_P z$ then $x \leq_P z$ (transitivity).

The pair $(P, \leq_P)$ is called a *partially ordered set* or *poset* for short. For a pair of elements $x, y \in P$ such that $x \leq_P y$ we say that x *precedes* y and y *succeeds* x. Elements $x, y \in P$ are called *comparable* if either $x \leq_P y$ or $y \leq_P x$; otherwise they are called *incomparable*. We will identify the poset with the elements P, assuming that the underlying order $\leq_P$ is fixed.

We define two other relations that can equivalently represent the partial order. First, let $<_P$ be the strict variant of $\leq_P$, that is, for all $x, y \in P$ we have $x <_P y$ if and only if $x \leq_P y$ and $x \neq y$. Second, let $\prec_P$ be the transitive reduction of $<_P$, that is, for all $x, y \in P$ we have $x \prec_P y$ if and only if $x <_P y$ and there is no $z \in P$ such that $x <_P z <_P y$. The relation $\prec_P$ is commonly known as the *cover relation*. A poset can be visualized as a DAG, where each vertex corresponds to an element in P and an arc between vertices corresponds to a pair in one of the relations (Figure. 5.1). A particularly concise representation is the *Hasse diagram*, where the arcs correspond to the cover relation and the elements are arranged so that all arcs point upwards, allowing one to omit the arrowheads.

A subset of elements $A \subseteq P$ is a *chain* if all pairs of elements in A are comparable. Conversely, a subset $A \subseteq P$ is an *antichain* if no pairs of elements in A are comparable. The *height* of P is the size of the largest chain in P, while the *width* of P, denoted $w(P)$, is the size of the largest antichain. A *downset* is a set $D \subseteq P$ such that if $y \in D$ and $x \leq_P y$ then $x \in D$. Dually, an *upset* is a set $U \subseteq P$ such that if $x \in U$ and $x \leq_P y$ then $y \in U$. An element $x \in P$ is *minimal* if it has no predecessors, that is, there is no $y \in P$ such that $y \leq_P x$. Likewise, x is *maximal* if it has no successors, that is, there is no $y \in P$ such that $x \leq_P y$. All of these definitions remain equivalent if $\leq_P$ is replaced with $<_P$ or $\prec_P$.

If P is a chain, then the order relation $\leq_P$ is called a *total order* or *linear order* on P. Further, a linear order $\leq_\ell$ is called a *linear extension* of $\leq_P$ if $x \leq_P y$ implies $x \leq_\ell y$ for all $x, y \in P$. A linear extension can be equivalently represented as a bijection $\sigma : P \to [n]$ such that $n = |P|$ and $x \leq_P y$ implies $\sigma(x) \leq \sigma(y)$ for all $x, y \in P$. In particular, this characterization holds if $\leq_P$ is replaced by $\prec_P$, which we will be using in the latter part of this chapter.

We denote by $\ell(P)$ the number of linear extensions of P.

5.2 Counting by recursive decomposition

The task of counting linear extensions can be decomposed into subproblems in multiple ways. One such method follows from the basic observation that every linear extension of a nonempty poset P begins with some minimal element. By applying this observation recursively, we have that

$$\ell(P) = \sum_{x \in \min(P)} \ell(P \setminus x)\,, \tag{5.1}$$

where $\min(P)$ denotes the set of minimal elements of P and $P \setminus x$ denotes the poset obtained by removing the element x. Evaluating recurrence (5.1) directly is equivalent to enumerating linear extensions one by one and will thus require factorial time in the worst case. This evaluation involves computing the subproblem $\ell(U)$ for all upsets $U \subseteq P$, often multiple times. By caching these intermediate results one obtains an algorithm [74] running in $O(|\mathcal{U}| \cdot w)$ time, where $\mathcal{U}$ is the set of upsets of P, and thus in $O(2^n n)$ time in the worst case. By using Dilworth's theorem [31] we can also show that $|\mathcal{U}| = O(n^w)$, making the algorithm fast for posets of low width but very inefficient when the graph representation of the poset is sparse.

We next observe that when recurrence (5.1) is applied to a sparse poset, many of the subproblems turn out to be disconnected, which allows one to

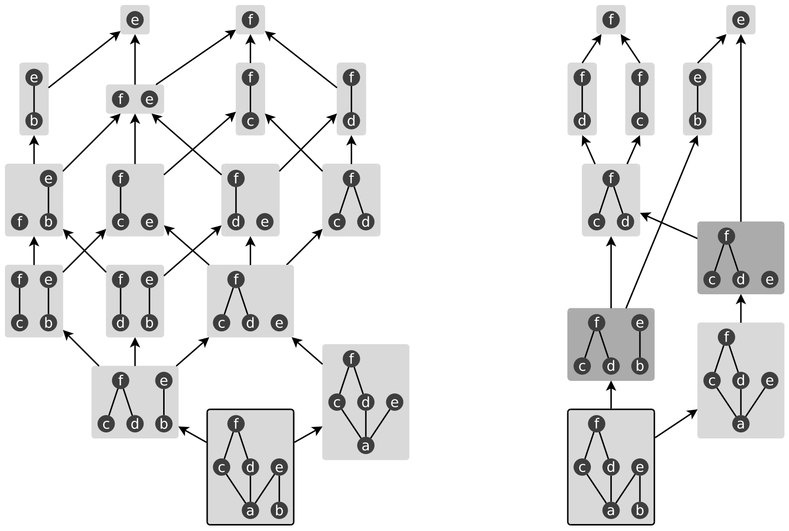


Figure 5.2: A poset on six elements (a–f) split into subproblems recursively. Left: All subproblems are solved by applying recurrence (5.1). Right: Disconnected subproblems (dark grey) are solved by applying recurrence (5.2).

solve them faster. Specifically, whenever P can be partitioned into sets A and B such that a and b are incomparable for all $a \in A$ and $b \in B$, we have that

$$\ell(P) = \ell(A) \cdot \ell(B) \cdot \binom{|P|}{|A|}. \tag{5.2}$$

Applying recurrence (5.2) whenever possible and recurrence (5.1) otherwise may significantly reduce the number of subproblems that need to be solved, as illustrated in Figure 5.2. Recurrence (5.2) can also be generalized for more than two components, in which case for each connected upset the algorithm will consider at most w upsets that are not connected. The total number of subproblems is thus at most $|\mathcal{U} \cap \mathcal{C}| \cdot (w + 1)$, where $\mathcal{C}$ denotes the set of connected sets of P, that is, subsets to which recurrence (5.2) is not applicable. We next discuss a variant of the first recurrence that may further reduce the number of subproblems.

Consider briefly the algorithm that only applies recurrence (5.1). By a symmetrical argument, the recurrence still holds if we have x run over the *maximal* elements of P instead, in which case the subproblems we need to solve are the *downsets* of P. Since the complement of an upset is a downset and vice versa, the number of subproblems and the running time of the algorithm remain unchanged. With the addition of recurrence (5.2)

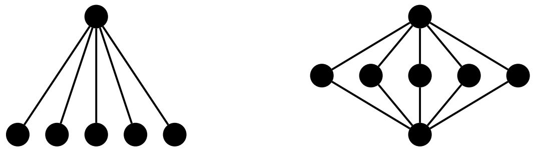


Figure 5.3: Left: A poset where all upsets are connected. Right: A poset where all upsets *and* downsets are connected.

this no longer holds, however, as the complement does not in general preserve connectivity. To exemplify, consider the simple poset in Figure 5.3 (left), where removing minimal elements will never make recurrence (5.2) applicable, whereas removing the lone maximal element splits the remainder into singletons.

In general, it is non-trivial to decide which variant of recurrence (5.1) should be used on a given poset. In some cases (Figure 5.3, right) it is clearly beneficial to alternate the choice depending on the subproblem. On the other hand, this also breaks the property that the number of subproblems is always bounded by the number of upsets and might thus increase the time requirement. Indeed, preliminary experiments for Paper V suggested that on most posets one should make the choice once only and then apply it consistently to every subproblem. Two heuristics for making this decision are proposed, one based on the number of minimal and maximal elements only and another that aims to estimate the number of resulting subproblems. Paper V shows that in practice both heuristics turn out to almost always make the better choice on randomly generated posets.

Other recursive techniques for counting linear extensions include the so-called *admissible partitions* and *static sets*. Following Paper V, we will call these Rule 2 and Rule 3, respectively, while the recurrences (5.1) and (5.2) are called Rule 1 and Rule 4. We will refer to all algorithms as the combination of rules they apply; hence, the two algorithms introduced so far are denoted R1 and R14. Other algorithms include R24 [92] and R34 [72], where the latter falls back into raw enumeration when neither rule is applicable and can therefore be immediately improved to R134. Experiments presented in Paper V show that R14 equipped with the simple heuristic outperforms the other algorithms by a large margin on a wide range of randomly generated posets (Figure 5.4). They also demonstrate the significance of making a good choice between the two variants of recurrence (5.1).

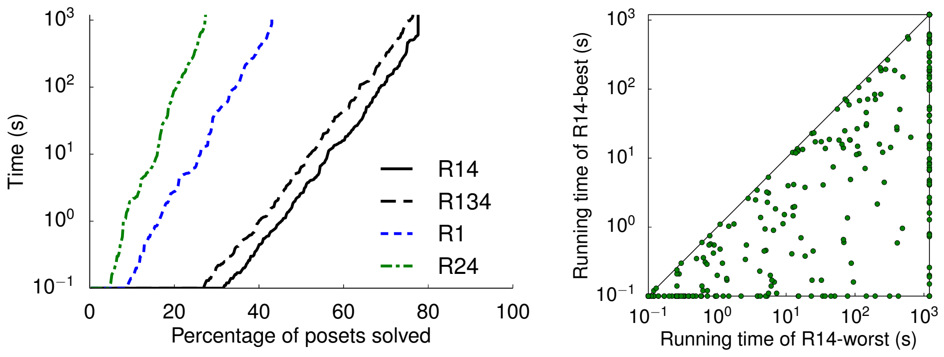


Figure 5.4: Experimental results on counting linear extensions by recursive decomposition. Left: The percentage of problem instances (posets) that were solved by each algorithm within a certain amount of time. The performance of R14 and R134 was evaluated using the simple heuristic for choosing between the two variants of recurrence (5.1). Right: A comparison between R14-best and R14-worst, which illustrate the best and worst possible running time of R14 in the hypothetical event that the heuristic always makes a good or bad choice, respectively.

5.3 Counting by variable elimination

In the past few decades it has been discovered that several problems that are NP-hard in the general case become tractable when an associated graph has bounded treewidth. Such results typically stem from a general method of nonserial dynamic programming [7], which has been rediscovered and reformulated several times [98, 30] under different names such as *variable elimination* [60, Ch. 9]. Briefly, consider a sum-product problem of form

$$\sum_{x_1,\ldots,x_n} \prod_{f_S \in F} f_S(x_S)\,,$$

where each variable x_i runs over a finite set of values and each $f_S \in F$ is a real-valued function that depends on a subset $S \subseteq \{1,\ldots,n\}$ of the variables. We associate such a problem with an *interaction graph*, an undirected graph where the variables are vertices and two variables are joined by an edge if and only if some $f_S \in F$ depends on both of them. With variable elimination the sum-product and many similar problems can be solved in time that is polynomial in n and exponential only in the treewidth t of the interaction graph. This assumes an optimal tree decomposition on the graph, which can be found in $O(n^{t+2})$ time [2].

Many important problems such as inference in graphical models [69, 30] can be expressed as a sum-product and thus solved by variable elimination. In particular, we aim to formulate the problem of counting linear extensions as a sum-product problem, with the intent that the summation ranges over all permutations σ of the elements, and the product equals either 1 or 0, depending on whether σ is a valid linear extension.

To that end, let P be a poset and consider an arbitrary mapping $\sigma : P \to [n]$. Recall that σ is a linear extension of P if and only if it is bijective and respects the cover relation $\prec_P$. To characterize such σ, we define the product

$$\Phi(\sigma) = \prod_{\substack{x,y \in P \\ x \prec_P y}} [\sigma_x < \sigma_y] \,.$$

For each pair (x, y) of the cover relation, the Iverson bracket $[\sigma_x < \sigma_y]$ evaluates to 1 if σ respects the order of the pair and otherwise to 0. Thus, for all bijections σ we have that $\Phi(\sigma)$ is an indicator function for whether σ is a linear extension. Therefore, we obtain the number of linear extensions by summing Φ over all bijections,

$$\ell(P) = \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma) \,. \tag{5.3}$$

The summation over σ can be equivalently expressed as a summation over variables $\sigma_1, \ldots, \sigma_n$, each σ_i taking values in $\{1, \ldots, n\}$. The only obstacle now is the requirement that σ must be a bijection. This imposes a global constraint on the variables, which the variable elimination scheme does not readily accommodate.

The principle of inclusion and exclusion allows us to write (5.3) as

$$\ell(P) = \sum_{k=1}^{n} \binom{n}{k} (-1)^{n-k} \sum_{\sigma : P \to [k]} \Phi(\sigma) \,,$$

thus removing the bijectivity constraint from the inner summation at the cost of having to compute it for all $k = 1, \ldots, n$. The inner sum is now a proper sum-product problem whose interaction graph (by definition of Φ) is exactly the undirected cover graph of P. With variable elimination the inner sum can be computed in $O(n^2 \cdot k^{t+1})$ time, which brings the total running time to $O(n^{t+4})$, where t is the treewidth of the cover graph. The details of the inclusion–exclusion step and the running time analysis are given in Paper V.

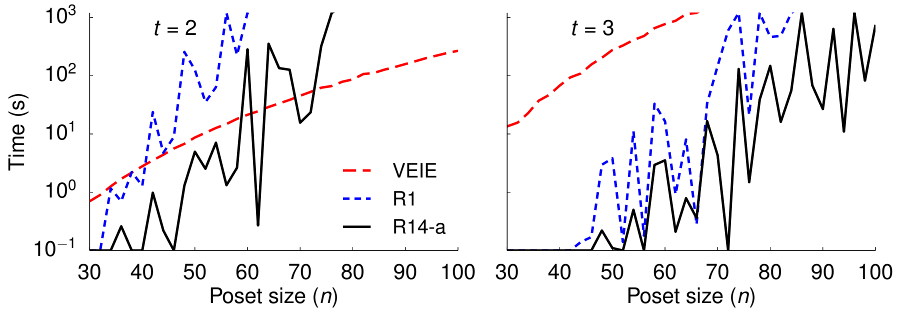


Figure 5.5: The behavior of variable elimination (VEIE) on graphs of treewidth 2 (left) and 3, compared to the recursive algorithms.

Paper V presents an experimental evaluation of the variable elimination algorithm (VEIE) on graphs of low treewidth, obtained by linking small grid posets along their sides and orienting the edges so that no directed cycles are introduced. Figure 5.5 shows that, while the recursive algorithms are often be faster in practice, the variable elimination algorithm eventually outperforms them in the worst case as the size n grows. In particular, since the recursive algorithms cache all subproblem solutions, their space usage often becomes prohibitively high for larger posets, whereas VEIE runs in polynomial space for bounded treewidth.

In the context of parameterized complexity [32], the running time of form $n^{f(t)}$ places the problem of counting linear extensions in the complexity class XP when parameterized by the treewidth of the cover graph, whereas a strictly better running time of form $f(t) \cdot n^{O(1)}$ would place the problem in the class FPT (fixed-parameter tractable). A recent complexity result [34] by the present author et al. complements the variable elimination algorithm by establishing that the problem is W[1]-hard when parameterized by the treewidth of the cover graph. Unless a common assumption W[1] $\neq$ FPT fails, this implies that the problem is not fixed-parameter tractable, though we may still hope to reduce the $t + 4$ exponent of the running time.

Chapter 6

Connected sets

Previous chapters have focused on the score-based approach to structure learning, with the approach based on independence tests briefly mentioned in Chapter 1. For Bayesian networks, a hybrid [93] of these two approaches aims to narrow down the search by first using independence tests to learn a *super-structure*, an undirected graph that contains an optimal structure with high confidence. An optimal structure is then found with a score-based search, restricted to DAGs whose undirected skeletons are subgraphs of the super-structure.

The running time of the score-based search phase is bounded by the number of so-called connected sets of the super-structure. Formally, in an undirected graph G with the vertex set V, a subset of vertices $U \subseteq V$ is called a *connected set* if the induced subgraph $G[U]$ is connected. Besides structure learning, the number of connected sets bounds the running time of algorithms for graph problems such as travelling salesman [11], maximum internal spanning tree [8], evaluation of the Tutte polynomial [10], and, by the result in Chapter 5, the problem of counting linear extensions. In this light, it is natural to ask if we can obtain upper bounds on the number of connected sets in different graph classes. Since an n-star already has $2^{n-1}+n$ connected sets, to improve upon the trivial $O(2^n)$ bound it appears necessary to restrict the maximum vertex degree of the graph.

In this chapter we study the number of connected sets in graphs of bounded vertex degree. For maximum degree $d \leq 2$, the strict upper bounds are trivial, attained by disjoint edges for $d = 1$ and by cycles for $d = 2$. For $d \geq 3$, an entropy lemma known as Shearer's inequality has yielded a general upper bound $\beta_d^n + n$, where $\beta_d = (2^{d+1} - 1)^{1/(d+1)}$ [11]. This is the best known bound for general d, and is suggested by empirical estimation [93] to be relatively loose. By considering the entropy lemma in an expanded context we will show improved upper bounds $b_d^n + n^{O(1)}$

Table 6.1: Upper and lower bounds on the maximum number of connected sets in graphs of degree d.

d	3	4	5
β_d	1.9680	1.9874	1.9948
b_d	1.9351	1.9812	1.9940
a_d	1.5537	1.6180	1.7320

in the case $d \in \{3, 4, 5\}$. Dually, we will prove respective lower bounds by constructing infinite families of graphs with at least a_d^n connected sets. Table 6.1 summarizes the values of a_d, b_d, and β_n. This chapter is based on Paper VI.

6.1 Upper bounds

Our main tool for obtaining upper bounds is Shearer's inequality [22], which relates the entropies of a joint random variable and its subsets. In a combinatorial context, this lemma allows us bound the size of a set family based on the sizes of its *projections*:

Lemma 6.1. *Let V be an n-element set with subsets $A_1, A_2, \ldots, A_k$, called* projectors*, such that each $v \in V$ appears in at least δ subsets. Let $\mathcal{F}$ be a family of subsets of V, and for each $i = 1, \ldots, k$ define the* projection *of $\mathcal{F}$ onto the projector A_i as $\mathcal{F}_i = \{F \cap A_i : F \in \mathcal{F}\}$. Then,*

$$|\mathcal{F}|^\delta \le \prod_{i=1}^{k} |\mathcal{F}_i| \,. \tag{6.1}$$

For the remainder of the section, consider an undirected graph G on the vertex set V. Let $n = |V|$ and let d be the largest vertex degree of G.

Our intent is to apply Lemma 6.1 to bound the number of connected sets of G. Evidently, the bound given by the lemma is useful only if we can also obtain a good bound on the size of each $\mathcal{F}_i$. To that end, it will be useful to let the projectors A_i be the vertex neighborhoods of G. As we will see, this allows us to prune out subsets that cannot be intersections between connected sets and A_i. For every $v \in V$ and $r = 0, 1, \ldots$, let $N^r[v]$ denote the *closed neighborhood of radius* r of v, that is, the set containing v and all $u \in V$ such that the shortest path between u and v uses at most

r edges. The size of $N^r[v]$ is bounded by the *Moore bound* δ_r, defined as

$$\delta_r = 1 + d\sum_{i=0}^{r-1}(d-1)^i = \frac{d(d-1)^r - 2}{d-2} \ .$$

For now, let $\mathcal{F}$ be an arbitrary set of subsets of V. We would like to apply Lemma 6.1 by using the neighborhoods $N^r[v]$ as the projector sets A_i for some fixed choice of r. To this end, let $\mathcal{F}_{v,r} = \{F \cap N^r[v] : F \in \mathcal{F}\}$ denote the projection of $\mathcal{F}$ into $N^r[v]$. Recall that the bound given by Lemma 6.1 depends crucially on δ, which we want to be as large as possible. In the best case all neighborhoods are of maximum size, and we can choose $\delta = \delta_r$. When this is not the case, we can still reach δ_r by augmenting the neighborhoods with additional vertices. Specifically, for each $v \in V$ define the projector A_v by first setting $A_v := N^r[v]$. Then, for each $u \in V$ that is contained in $k < \delta_r$ projectors, add u to $\delta_r - k$ projectors that do not already contain it (it does not matter which).

We now define for each $v \in V$ the projection $\mathcal{F}_v = \{F \cap A_v : F \in \mathcal{F}\}$. Each additional vertex at most doubles the size of $\mathcal{F}_v$ and it is thus sufficient to bound the size of $\mathcal{F}_{v,r}$. Suppose that we can give a uniform bound that rules out a fixed proportion of potential subsets, that is, for some $0 \le \rho \le 1$ we have that $|\mathcal{F}_{v,r}| \le 2^{|N^r[v]|}\rho$ for all $v \in V$. Then,

$$|\mathcal{F}_v| \le |\mathcal{F}_{v,r}| \cdot 2^{|A_v|-|N^r[v]|} \le 2^{|N^r[v]|}\rho \cdot 2^{|A_v|-|N^r[v]|} \le 2^{|A_v|}\rho \, .$$

and by Lemma 6.1 we have that

$$|\mathcal{F}| \le (2\rho^{1/\delta_r})^n \, . \tag{6.2}$$

Obtaining an upper bound for the size of $\mathcal{F}$ thus comes down to finding a good value for ρ.

The aforementioned $\beta_d^n + n$ bound follows by considering the case $r = 1$. Specifically, let $\mathcal{F}$ be the set connected sets excluding the n singleton sets $\{v\}$ for all $v \in V$. Since $N_v^1[v]$ contains all neighbors of v, we must have $F \cap N^r[v] \ne \{v\}$ for all $v \in V$ and $F \in \mathcal{F}$. As a result we may take $\rho = 1 - 1/2^{d+1}$, where $\delta_r = d + 1$, and the bound follows from (6.2).

Our intent is to improve on this bound by considering a radius $r \ge 2$. As above, it will be useful to exclude a polynomial number of special subsets from consideration. We say that a nonempty connected set is *local* if it is contained entirely in at least one of the neighborhoods. We observe that there are at most $(2^{\delta_r} - 1)n$ local connected sets and thus define $\mathcal{F}$ to be the set of connected sets that are not local. In order to find a good value for ρ, let us consider what feasible projections of $\mathcal{F}$ to vertex neighborhoods may

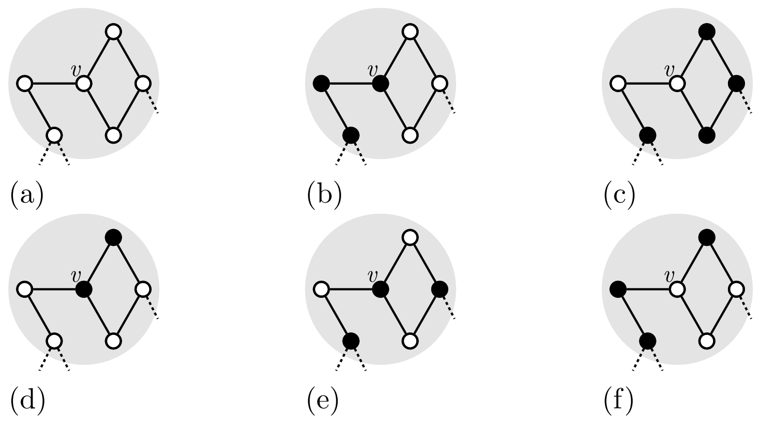


Figure 6.1: A possible neighborhood $N^2[v]$ in a graph of degree $d = 3$, and subsets of the neighborhood that can (a–c) or cannot (d–f) belong to the respective projection $\mathcal{F}_{v,2}$ of non-local connected sets. The dashed lines represent potential edges that may connect vertices at the boundary to vertices outside of the neighborhood.

look like. Consider an intersection U between a non-local connected set C and a neighborhood $N^r[v]$. Since C contains a vertex outside of $N^r[v]$, it must thus "exit" the neighborhood via a vertex $u \in U$ that has a neighbor outside of $N^r[v]$. To have such a neighbor, the vertex u must be exactly at distance r from v and may have at most $d - 1$ neighbors within $N^r[v]$. We say that such a vertex resides at the *boundary* of the neighborhood. Further, we say that U is *boundary-connected* if every connected component of $G[U]$ contains a vertex at the boundary. Figure 6.1 shows examples of neighborhood subsets that are and are not boundary-connected.

The number of boundary-connected sets bounds the size of the projection $\mathcal{F}_{v,r}$ and can be computed without looking outside of the neighborhood $N^r[v]$. To optimize the value of ρ, it is therefore sufficient to count the number of boundary-connected sets in all possible neighborhoods. For a fixed choice of d and r, the number of such neighborhoods is finite and the computation can be carried out with computer search. The details of this procedure are presented in Paper VI. For $r = 2$ and $d \in \{3, 4, 5\}$, the search yields the upper bounds $b_d^n + n^{O(1)}$, where the polynomial term accounts for the local connected sets. For greater values of r and d, however, the method appears not to improve the bounds.

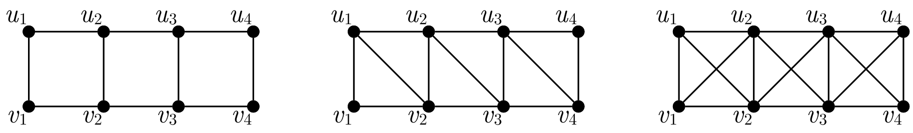


Figure 6.2: Ladder graphs with 8 vertices for degrees $d = 3, 4, 5$.

6.2 Lower bounds

It is natural to ask how far the improved bounds are from tight upper bounds on the number of connected sets. Obtaining corresponding lower bounds for graphs of bounded degree appears to be equally challenging. While empirical analysis [93] suggests that graphs having most connected sets are tree-like within neighborhoods of bounded radius, the global nature of connectedness makes analyzing such graphs highly non-trivial. We next provide lower bounds based on generalized ladder graphs, where the number of connected sets may be characterized recursively.

A *ladder graph* of length k and degree d contains $2k$ vertices, labelled $u_1, u_2, \ldots, u_k$ and $v_1, v_2, \ldots, v_k$, with exactly the following edges: First, u_i is adjacent to u_{i+1} and v_i is adjacent to v_{i+i} for all $1 \le i \le k-1$. Second, u_i is adjacent to v_j if and only if $0 \le i - j \le \lfloor (d-3)/2) \rfloor$ or $0 \le j - i \le \lceil (d-3)/2) \rceil$. Figure 6.2 shows examples of ladder graphs for $d \in \{3, 4, 5\}$. We analyze each of these cases separately.

Consider first a ladder graph of degree 3. For each $p = 1, 2, \ldots, k$, denote by $\mathcal{C}_p$ the set of connected sets that intersect $\{u_j, v_j\}$ for all $1 \le j \le p$. For our purposes, it will be sufficient to analyze the size of $\mathcal{C}_k$, as the number of other connected sets is within a polynomial factor from it. We will characterize the size of $\mathcal{C}_p$ inductively over p. To that end, partition each $\mathcal{C}_p$ into $\mathcal{S}_p$ and $\mathcal{T}_p$ such that every set in $\mathcal{S}_p$ contains exactly one of u_p and v_p, while every set in $\mathcal{T}_p$ contains both of them. Denote the size of $\mathcal{S}_p$ and $\mathcal{T}_p$ by s_p and t_p, respectively. We now have that $s_1 = 2$ and $t_1 = 1$, and for $p \ge 2$ we observe that $s_p = s_{p-1} + 2t_{p-1}$ and $t_p = s_{p-1} + t_{p-1}$. From these we can derive the homogeneous linear recurrence $s_p = 2s_{p-1} + s_{p-2}$, which has the solution

$$s_p = \frac{1}{\sqrt{2}}(1+\sqrt{2})^p - \frac{1}{\sqrt{2}}(1-\sqrt{2})^p \, .$$

A similar analysis for t_p yields that $|\mathcal{C}_k| = s_k + t_k \ge (1+\sqrt{2})^k - 1$ and therefore we get the lower bound $a_3 = (1+\sqrt{2})^{1/2} \simeq 1.5537$.

The analysis for $d = 4$ is very similar. We now let x_p, y_p, and z_p be the number of elements of $\mathcal{C}_p$ that contain, respectively, u_p but not v_p, v_p but not u_p, and both u_p and v_p. One now obtains the recurrences $x_p = x_{p-1} + z_{p-1}$, $y_p = x_{p-1} + y_{p-1} + z_{p-1}$, and $z_p = x_{p-1} + y_{p-1} + z_{p-1}$, and from their solutions $|\mathcal{C}_k| = x_k + y_k + z_k \geq (\frac{1}{2}(3 + \sqrt{5}))^k - 1$, implying the lower bound $a_4 = (\frac{1}{2}(3 + \sqrt{5}))^{1/2} \simeq 1.6180$.

The case $d = 5$ is comparatively easy. Simply observe that $C \in \mathcal{C}_p$ if and only if for all $i = 1, 2, \ldots, p$ it holds that C contains at least one of u_i and v_i. Thus, $|\mathcal{C}_k| = 3^k$, giving us the lower bound $a_5 = 3^{1/2} \simeq 1.7320$.

Chapter 7

Discussion

This thesis studied combinatorial problems that arise in structure learning in two families of graphical models, the Bayesian networks and the decomposable models. Papers I–VI gave new algorithms for such problems, as well as provided both empirical and analytical results on the performance of existing approaches.

Papers I and II considered Bayesian learning in decomposable models. By formulating the problem as dynamic programming over a recursive decomposition of junction trees, Paper I presented the first algorithm that is guaranteed to discover an optimal structure in at most exponential time. Paper II adapted the algorithm for computing posterior expectations but could only solve the problem for a specific non-modular structure prior. As a partial remedy, Paper II further turned the approach into a sampling algorithm that, via importance sampling, enables unbiased estimation of posterior expectations for modular and other priors. Papers I and II established the applicability of these techniques for instances of moderate size (up to around 20 variables). The algorithms run in $O(4^n)$ time, which is in sharp contrast with the $O(2^n n^2)$ time algorithms for learning Bayesian networks. A major open question is whether the running time can be reduced, or whether structure learning is indeed inherently harder in decomposable models, despite the fact that they are a strictly more constrained model class. Unlike in Bayesian networks, it is also not clear whether there exists an efficient way to prune the input scores or to employ A* style methods to avoid visiting the entire search space. Other challenges for future work include reducing the space complexity, possibly in exchange for time, as well as obtaining controllable approximation guarantees on the estimates of posterior expectations.

Papers III and IV studied the empirical hardness of structure learning in Bayesian networks. As a starting point, the papers demonstrated that the performance of state-of-the-art exact algorithms varies significantly on per-instance basis, thus raising the problem of algorithm selection. To tackle the problem, supervised learning of running time predictors for the algorithms was considered. In particular, the papers showed that (i) even simple features of problem instances are sufficient to yield a portfolio that accurately determines the fastest algorithm on a given instance, and (ii) incorporating more sophisticated features significantly improves prediction accuracy. An immediate challenge for future work is to identify more useful features and close the remaining gap between current and optimal portfolio behavior. A potential further step is to leverage the hardness models to develop novel search techniques that go beyond the current state-of-the-art, possibly by employing a divide-and-conquer method and using the models to choose appropriate algorithms for different subproblems.

Paper V studied the classic problem of counting linear extensions of partially ordered sets, which arises in various contexts such as bias correction for sampling Bayesian networks. The paper presented two algorithms for the problem, based on recursive decomposition into subproblems and the method of variable elimination, respectively. Both algorithms target sparse orders in particular, and their practical efficiency was demonstrated by experiments. Whether the first algorithm can be improved further is dependent on the discovery of novel decomposition rules and efficient ways to determine when they are applicable. For the second algorithm, the problem had to be transformed via a step of inclusion–exclusion in order to make variable elimination applicable. It remains an open question whether there exists a more straightforward formulation that is equally or more efficient but avoids this complication.

Finally, Paper VI studied the extremal combinatorics of connected sets, whose number bounds the running time of algorithms for structure learning and other problems. Specifically, the paper generalized and seemingly exhausted an entropy method for deriving upper bounds on the number of connected sets in graphs of bounded degree. This resulted in a modest improvement in previously known bounds, which was complemented by lower bounds derived from a simple family of graphs. There remains a considerable gap between current upper and lower bounds, and narrowing it down remains a task for future research.

References

[1] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[2] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[3] Mike D. Atkinson. On computing the number of linear extensions of a tree. *Order*, 7(1):23–25, 1990.

[4] Mark Bartlett and James Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, In press, 2015.

[5] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

[6] Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 86–95. JMLR.org, 2014.

[7] Umberto Bertelè and Francesco Brioschi. *Nonserial dynamic programming.* Mathematics in science and engineering. Academic Press, New York, 1972.

[8] Daniel Binkele-Raible, Henning Fernau, Serge Gaspers, and Mathieu Liedloff. Exact and parameterized algorithms for max internal spanning tree. *Algorithmica*, 65(1):95–128, 2013.

[9] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceed-*

ings of the 39th Annual ACM Symposium on Theory of Computing (STOC), pages 67–74, 2007.

[10] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 677–686. IEEE Computer Society, 2008.

[11] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Transactions on Algorithms*, 8(2):18:1–13, 2012.

[12] Andreas Björklund, Mikko Koivisto, Thore Husfeldt, Jesper Nederlof, Petteri Kaski, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1436–1444, 2012.

[13] Remco R. Bouckaert. Probalistic network construction using the minimum description length principle. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, volume 747 of *Lecture Notes in Computer Science*, pages 41–48. Springer, 1993.

[14] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees.* Chapman and Hall/CRC, 1984.

[15] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.

[16] Russ Bubley and Martin E. Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201(1-3):81–88, 1999.

[17] Wray Buntine. Theory refinement on Bayesian networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 52–60. Morgan Kaufmann Publishers Inc., 1991.

[18] Jie Cheng, Russell Greiner, Jonathan Kelly, David A. Bell, and Weiru Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90, 2002.

[19] David M. Chickering. *Learning from Data: Artificial Intelligence and Statistics V*, chapter Learning Bayesian Networks is NP-Complete, pages 121–130. Springer, 1996.

[20] David M. Chickering, Dan Geiger, and David Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 112–128, 2003.

[21] David M. Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[22] Fan R. K. Chung, Ronald L. Graham, Peter Frankl, and James B. Shearer. Some intersection theorems for ordered sets and graphs. *Journal of Combinatorial Theory, Series A*, 43(1):23–37, 1986.

[23] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[24] Jukka Corander, Mats Gyllenberg, and Timo Koski. Bayesian model learning based on a parallel MCMC strategy. *Statistics and Computing*, 16(4):355–362, 2006.

[25] Jukka Corander, Tomi Janhunen, Jussi Rintanen, Henrik Nyman, and Johan Pensar. Learning chordal Markov networks by constraint satisfaction. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 1349–1357. Curran Associates, Inc., 2013.

[26] James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 153–160. AUAI Press, 2011.

[27] James Cussens and Mark Bartlett. Advances in Bayesian network learning using integer programming. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 182–191. AUAI Press, 2013.

[28] Alexander P. Dawid and Steffen. L. Lauritzen. Hyper Markov laws in the statistical analysis of decomposable graphical models. *The Annals of Statistics*, 21(3):1272–1317, 1993.

[29] Cassio P. de Campos and Qiang Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

[30] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[31] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[32] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 2012.

[33] Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991.

[34] Eduard Eiben, Robert Ganian, Kustaa Kangas, and Sebastian Ordyniak. Counting linear extensions: Parameterizations by treewidth. In *24th Annual European Symposium on Algorithms (ESA)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[35] Xiannian Fan, Brandon M. Malone, and Changhe Yuan. Finding optimal Bayesian network structures with constraints learned from data. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 200–209. AUAI Press, 2014.

[36] Stefan Felsner and Thibault Manneville. Linear extensions of N-free orders. *Order*, 32(2):147–155, 2015.

[37] Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1-2):95–125, 2003.

[38] Paolo Giudici and Robert Castelo. Improving Markov Chain Monte Carlo model search for data mining. *Machine Learning*, 50(1-2):127–158, 2003.

[39] Paolo Giudici and Peter J. Green. Decomposable graphical Gaussian model determination. *Biometrika*, 86(4):785–801, 1999.

[40] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[41] Peter J. Green and Alun Thomas. Sampling decomposable graphs using a Markov chain on junction trees. *Biometrika*, 100(1):91–110, 2013.

[42] Marco Grzegorczyk and Dirk Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71(2-3):265–305, 2008.

[43] Michel Habib and Rolf H. Möhring. On some complexity properties of N-free posets and posets with bounded decomposition diameter. *Discrete Mathematics*, 63(2-3):157–182, 1987.

[44] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[45] David Heckerman. A tutorial on learning with Bayesian networks. In *Innovations in Bayesian Networks: Theory and Applications*, volume 156 of *Studies in Computational Intelligence*, pages 33–82. Springer, 2008.

[46] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[47] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 16th ACM National Meeting*, ACM '61, pages 71.201–71.204. ACM, 1961.

[48] Holger Hoos, Marius T. Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, 14(4-5):569–585, 2014.

[49] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[50] Tommi S. Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *JMLR Proceedings*, pages 358–365. JMLR.org, 2010.

[51] Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer, 1st edition, 2010.

[52] Kustaa Kangas, Teemu Hankala, Teppo Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 603–609. IJCAI/AAAI Press, 2016.

[53] Kustaa Kangas, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. On the number of connected sets in bounded degree graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 8747 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2014.

[54] Kustaa Kangas, Mikko Koivisto, and Teppo Niinimäki. Learning chordal Markov networks by dynamic programming. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 2357–2365. Curran Associates, Inc., 2014.

[55] Kustaa Kangas, Teppo Niinimäki, and Mikko Koivisto. Averaging of decomposable graphs by dynamic programming and sampling. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 415–424. AUAI Press, 2015.

[56] Robert Kennes and Philippe Smets. Computational aspects of the Mobius transformation. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 401–416, 1990.

[57] Mikko Koivisto. Advances in exact Bayesian structure discovery in Bayesian networks. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI).* AUAI Press, 2006.

[58] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[59] Kaname Kojima, Eric Perrier, Seiya Imoto, and Satoru Miyano. Optimal search on clustered structural constraint for learning Bayesian network structure. *Journal of Machine Learning Research*, 11:285–310, 2010.

[60] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques.* MIT Press, 2009.

[61] Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 370–378. JMLR.org, 2013.

[62] Timo Koski and John Noble. A review of Bayesian networks and structure learning. *Mathematica Applicanda (Matematyka Stosowana)*, 40(1):51–103, 2012.

[63] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[64] Lars Kotthoff, Ian P. Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270, 2012.

[65] Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *Revised Selected Papers of the Ninth International Conference on Learning and Intelligent Optimization (LION)*, volume 8994 of *Lecture Notes in Computer Science*, pages 202–217. Springer, 2015.

[66] Wai Lam and Fahiem Bacchus. Using causal information and local measures to learn Bayesian networks. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 243–250. Morgan Kaufmann, 1993.

[67] Pedro Larrañaga, Mikel Poza, Yosu Yurramendi, Roberto H. Murga, and Cindy M. H. Kuijpers. Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(9):912–926, 1996.

[68] Steffen L. Lauritzen. *Graphical models.* Oxford statistical science series. Clarendon Press, 1996.

[69] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[70] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *Lecture Notes in Computer Science*, pages 556–572. Springer, 2002.

[71] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4), 2009.

[72] Wing-Ning Li, Zhichun Xiao, and Gordon Beavers. On computing the number of topological orderings of a directed acyclic graph. *Congressus Numerantium*, 174:143–159, 2005.

[73] Moshe Lichman. UCI machine learning repository, 2013. University of California, Irvine, School of Information and Computer Sciences.

[74] Karel De Loof, Hans De Meyer, and Bernard De Baets. Exploiting the lattice of ideals representation of a poset. *Fundamenta Informaticae*, 71(2-3):309–321, 2006.

[75] Thomas Lukasiewicz, Maria V. Martinez, and Gerardo I. Simari. Probabilistic preference logic networks. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 561–566. IOS Press, 2014.

[76] David Madigan, Jeremy York, and Denis Allard. Bayesian graphical models for discrete data. *International Statistical Review / Revue Internationale de Statistique*, 63(2):215–232, 1995.

[77] Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Empirical hardness of Bayesian network structure learning. Under revision.

[78] Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Predicting the hardness of learning Bayesian networks. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2460–2466. AAAI Press, 2014.

[79] Heikki Mannila and Christopher Meek. Global partial orders from sequential data. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 161–168. ACM, 2000.

[80] Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 403–410. Morgan Kaufmann, 1995.

[81] Rolf H. Möhring. *Algorithms and Order*, chapter Computationally tractable classes of ordered sets, pages 105–193. Springer, 1989.

[82] Jason Morton, Lior Pachter, Anne Shiu, Bernd Sturmfels, and Oliver Wienand. Convex rank tests and semigraphoids. *SIAM Journal on Discrete Mathematics*, 23(3):1117–1134, 2009.

[83] Christian J. Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, 57:113–149, 2016.

[84] Teppo M. Niinimäki and Mikko Koivisto. Annealed importance sampling for structure learning in Bayesian networks. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1579–1585. IJCAI/AAAI, 2013.

[85] Sascha Ott, Seiya Imoto, and Satoru Miyano. Finding optimal models for small gene networks. In *Proceedings of the Pacific Symposium on Biocomputing 2004*, pages 557–567. World Scientific, 2004.

[86] Pekka Parviainen, Hossein Shahrabi Farahani, and Jens Lagergren. Learning bounded tree-width Bayesian networks using integer linear programming. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 751–759. JMLR.org, 2014.

[87] Pekka Parviainen and Mikko Koivisto. Exact structure discovery in Bayesian networks with less space. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 436–443. AUAI Press, 2009.

[88] Pekka Parviainen and Mikko Koivisto. Bayesian structure discovery in Bayesian networks with less space. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *JMLR Proceedings*, pages 589–596. JMLR.org, 2010.

[89] Pekka Parviainen and Mikko Koivisto. Finding optimal Bayesian networks using precedence constraints. *Journal of Machine Learning Research*, 14(1):1387–1415, 2013.

[90] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers Inc., 1988.

[91] Judea Pearl and Thomas Verma. A theory of inferred causation. In *Proceedings of the Second International Conference on Principles*

of Knowledge Representation and Reasoning (KR), pages 441–452. Morgan Kaufmann, 1991.

[92] Marcin Peczarski. New results in minimum-comparison sorting. *Algorithmica*, 40(2):133–145, 2004.

[93] Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal Bayesian network given a super-structure. *Journal of Machine Learning Research*, 9:2251–2286, 2008.

[94] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, 2nd Edition.* Cambridge University Press, 1992.

[95] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[96] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[97] Paul Saikko, Brandon Malone, and Matti Järvisalo. MaxSAT-based cutting planes for learning graphical models. In *12th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, volume 9075 of *Lecture Notes in Computer Science*, pages 347–356. Springer, 2015.

[98] Ross D. Shachter, Bruce D'Ambrosio, and Brendan Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 126–131. AAAI Press / The MIT Press, 1990.

[99] Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 445–452. AUAI Press, 2006.

[100] Tomi Silander, Teemu Roos, Petri Kontkanen, and Petri Myllymäki. Factorized normalized maximum likelihood criterion for learning Bayesian network structures. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models (PGM)*, pages 257–264, 2008.

[101] Ajit Singh and Andrew Moore. Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University, 2005.

[102] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search.* The MIT Press, second edition, 2001.

[103] Peter Spirtes and Christopher Meek. Learning Bayesian networks with discrete variables from data. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 294–299. AAAI Press, 1995.

[104] Nathan Srebro. Maximum likelihood bounded tree-width Markov networks. *Artificial Intelligence*, 143(1):123–138, 2003.

[105] Milan Studený and James Cussens. The chordal graph polytope for learning decomposable models. In *Proceedings of the Eighth International Conference on Probabilistic Graphical Models (PGM)*, volume 52 of *JMLR Workshop and Conference Proceedings*, pages 499–510. JMLR.org, 2016.

[106] Joe Suzuki. A construction of Bayesian networks from databases based on an MDL principle. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 266–273. Morgan Kaufmann, 1993.

[107] Yoshinori Tamada, Seiya Imoto, and Satoru Miyano. Parallel algorithm for learning optimal Bayesian network structure. *Journal of Machine Learning Research*, 12:2437–2459, 2011.

[108] Claudia Tarantola. MCMC model determination for discrete graphical models. *Statistical Modelling*, 4(1):39–61, 2004.

[109] Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 548–549. AUAI Press, 2005.

[110] Alun Thomas and Peter J. Green. Enumerating the junction trees of a decomposable graph. *Journal of Computational and Graphical Statistics*, 18(4):930–940, 2009.

[111] Jin Tian and Ru He. Computing posterior probabilities of structural features in Bayesian networks. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 538–547. AUAI Press, 2009.

[112] Peter van Beek and Hella-Franziska Hoffmann. Machine learning of Bayesian networks using constraint programming. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, volume 9255 of *Lecture Notes in Computer Science*, pages 429–445. Springer, 2015.

[113] Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the Eighth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 323–330. Morgan Kaufmann, 1992.

[114] Michael Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17:972–975, 1991.

[115] Chris S. Wallace, Kevin B. Korb, and Honghua Dai. Causal discovery via MML. In *Proceedings of the 13th International Conference on Machine Learning (ICML)*, pages 516–524. Morgan Kaufmann, 1996.

[116] Yong Wang and Ian H. Witten. Inducing model trees for continuous classes. In *Proceedings of the Ninth European Conference on Machine Learning Poster Papers*, pages 128–137, 1997.

[117] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[118] Frank Yates. *The design and analysis of factorial experiments.* Imperial Bureau of Soil Science. Harpenden, 1937.

[119] Changhe Yuan and Brandon M. Malone. An improved admissible heuristic for learning optimal Bayesian networks. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 924–933. AUAI Press, 2012.

[120] Changhe Yuan and Brandon M. Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

[121] Changhe Yuan, Brandon M. Malone, and XiaoJian Wu. Learning optimal Bayesian networks using A* search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2186–2191. IJCAI/AAAI, 2011.